

# Real-Time Monte Carlo Tree Search in *Ms Pac-Man*

Tom Pepels, Mark H. M. Winands, *Member, IEEE*, and Marc Lanctot

**Abstract**—In this paper, Monte Carlo tree search (MCTS) is introduced for controlling the Pac-Man character in the real-time game *Ms Pac-Man*. MCTS is used to find an optimal path for an agent at each turn, determining the move to make based on the results of numerous randomized simulations. Several enhancements are introduced in order to adapt MCTS to the real-time domain. *Ms Pac-Man* is an arcade game, in which the protagonist has several goals but no conclusive terminal state. Unlike games such as *Chess* or *Go* there is no state in which the player wins the game. Instead, the game has two subgoals, 1) surviving and 2) scoring as many points as possible. Decisions must be made in a strict time constraint of 40 ms. The Pac-Man agent has to compete with a range of different ghost teams, hence limited assumptions can be made about their behavior. In order to expand the capabilities of existing MCTS agents, four enhancements are discussed: 1) a variable-depth tree; 2) simulation strategies for the ghost team and Pac-Man; 3) including long-term goals in scoring; and 4) reusing the search tree for several moves with a decay factor  $\gamma$ . The agent described in this paper was entered in both the 2012 World Congress on Computational Intelligence (WCCI'12, Brisbane, Qld., Australia) and the 2012 IEEE Conference on Computational Intelligence and Games (CIG'12, Granada, Spain) Pac-Man Versus Ghost Team competitions, where it achieved second and first places, respectively. In the experiments, we show that using MCTS is a viable technique for the Pac-Man agent. Moreover, the enhancements improve overall performance against four different ghost teams.

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

## I. INTRODUCTION

**M**S PAC-MAN is a real-time arcade game based on the popular *Pac-Man* game released in 1980. The player controls the main character named Ms Pac-Man (henceforth named Pac-Man) through a maze, eating pills, and avoiding or chasing the four hostile ghosts. This paper discusses enhancements to the Monte Carlo tree search (MCTS) [1], [2] framework to allow its use in real-time, stochastic domains such as *Ms Pac-Man*.

Manuscript received May 21, 2013; revised September 10, 2013; accepted October 22, 2013. Date of publication February 04, 2014; date of current version September 11, 2014. This work was supported in part by Maastricht Research Based Learning (MARBLE) and by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature under Grant 612.000.938.

The authors are with the Games and AI Group, Department of Knowledge Engineering, Faculty of Humanities, Maastricht University, Maastricht 6200 MD, The Netherlands (e-mail: tom.pepels@maastrichtuniversity.nl; m.winands@maastrichtuniversity.nl; marc.lanctot@maastrichtuniversity.nl).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCIAIG.2013.2291577

MCTS has been successful when applied to several types of domains such as puzzles, card games, and board games [3]. However, research on real-time domains has been limited in comparison. MCTS has been applied to tactical assault planning in real-time strategy games [4], steering in the physical traveling salesman problem [5], and real-time hospital patient admissions planning [6]. These domains have unique characteristics that make planning more challenging than in the standard turn-based games. For example, computation time for search is often severely limited, the action spaces are large if not infinite, the state spaces are often continuous, and payoffs are sometimes not classified as either wins or losses.

In *Ms Pac-Man*, the player's goal is to avoid being eaten by the ghosts and gaining higher scores by collecting pills. *Ms Pac-Man* inherited its game mechanics from the original *Pac-Man*. Moreover, it introduced four different mazes, and more importantly, unpredictable ghost behavior. *Ms Pac-Man* has become an interesting subject for AI research. The game rules are straightforward, however complex planning and foresight are required for a player to achieve high scores. Additionally, the organization of recurring competitions provides a testbed to compare performance between different methods.

In 2007, the first *Ms Pac-Man* competition was organized, the *Ms Pac-Man* Competition (screen-capture version) [7]. In this competition, Pac-Man agents play the original version of the game in an emulator. Agents interpret a capture of the screen to determine the game state, and at each turn moves are passed to the emulator running the game. Games are played by a single Pac-Man agent, competing to get the highest average score over a number of games, whereas the ghosts behave according to a fixed, known strategy.

In 2011, a second competition was introduced, the *Ms Pac-Man* Versus Ghosts Competition [8]. This competition is played in a newly developed implementation of the game, in which the game state is fully available. Furthermore, Pac-Man agents compete with a variety of ghost teams also entered in the competition. The Pac-Man agents compete to get the highest scores, whereas the ghost teams compete for the lowest Pac-Man scores.

Although most Pac-Man agents entering these competitions are rule based, research has been performed on using general techniques, such as genetic programming [9], neural networks [10], and search trees [11]. Due to the successful application of MCTS in other games [3], interest in developing MCTS agents for *Ms Pac-Man* has grown. Samothrakis *et al.* [12] developed an MCTS agent using a  $\text{Max}^n$  search tree with score keeping for both Pac-Man and the four ghosts separately. Their method sets a target location in the current maze as a long-term goal for Pac-Man while MCTS computes the optimal route to the target in order to determine the best move. Other Monte-Carlo-based

agents were developed for achieving specific goals in *Ms Pac-Man*, such as ghost avoidance [13] and endgame situations [14], demonstrating the potential of Monte Carlo methods for Pac-Man agents. In 2011, the first MCTS agent won the Ms Pac-Man screen-capture competition [7]. Until then, rule-based agents led the competition. The victorious MCTS agent, NOZOMU [15], was designed to avoid so-called “pincer moves,” in which every escape path for Pac-Man is blocked. The approach was successful in beating the leading rule-based agent ICE PAMBUSH [16] with a high score of 36 280. In the same year, the first MCTS-based agents entered the Ms Pac-Man Versus Ghosts Competition. Both a ghost team [17], [18] and a Pac-Man agent [19] based on a combination of knowledge rules and MCTS search entered the competition. Moreover, the MCTS-based ghost team NQKIEN [17], [20], [21] won the 2011 Congress on Evolutionary Computation (CEC, New Orleans, LA, USA) [22] edition of the competition with an average of 11 407 points scored by its opponents.

This paper discusses the Pac-Man agent MAASTRICHT, which won the 2012 IEEE Conference on Computational Intelligence and Games (CIG’12, Granada, Spain) Ms Pac-Man Versus Ghosts Competition. The agent has full access to the game state and competes with a range of different ghost teams. The goal of this research is to develop a generally strong playing Pac-Man agent based on MCTS. To employ the MCTS framework for Pac-Man four enhancements are introduced: 1) a variable-depth tree; 2) simulation strategies for the ghost team and Pac-Man; 3) including long-term goals in scoring; and 4) reusing the search tree over several moves. This paper is an extension to the CIG’12 conference paper [23] published in 2012. Foremost it focuses on dealing with the real-time aspects of the game. We introduce a strategy to reuse the search tree in order to increase the number of simulations on which to base decisions. Moreover, enhancements to the simulation strategy and long-term goals are detailed, and more extensive experiments are performed against different ghost teams.

The paper is structured as follows. First, the *Ms Pac-Man* domain is introduced. Next, MCTS and the UCT selection policy are described, and enhancements to the MCTS framework discussed in detail. Finally, experimental and competition results are presented and a conclusion drawn.

## II. MS PAC-MAN

The rules of *Ms Pac-Man* are based on the classic arcade game. Pac-Man initially has three lives, which she loses through contact with a nonedible ghost. After losing a life due to being captured by a ghost, the location of all the ghosts and Pac-Man are reset to their initial configuration. The game environment consists of four different mazes, which are cycled alternately through the levels. The game progresses every time unit of 40 ms, forcing Pac-Man to make a move, i.e., up, down, left, or right. Pac-Man cannot explicitly skip a turn; it is, however, possible to “wait” by moving onto a wall, or reversing the previous move every turn. Ghosts are only allowed to make a move at a junction in the maze. On a path between junctions they can only travel forward. When a power pill is eaten by Pac-Man the ghosts become edible for a limited time: their movement speed decreases and they are instantly forced to reverse their direction.

The moment Pac-Man reaches a score of 10 000 by eating pills and ghosts, she gains a life.

Both the ghost team and Pac-Man have one in-game time unit to compute a move. Once computed and submitted, the ghost team’s and Pac-Man’s moves are executed simultaneously. If no move is submitted, a randomly selected one is executed. Furthermore, at each turn, a “global reverse” can occur with a small probability of 0.0015 where each ghost is immediately forced to reverse its direction. Pac-Man gains ten points for each pill eaten, 50 points per power pill, and 200<sup>n</sup> points for each ghost eaten, where  $n \in \{1, 2, 3, 4\}$  is the total number of ghosts eaten after taking the power pill. The game either ends naturally when Pac-Man has no lives remaining or after a set time limit.

The version of the game used in the Ms Pac-Man Versus Ghosts Competition is a two-player, fully observable, asymmetric, and stochastic game. This is in contrast with the original version of the game, which was single player, because the ghosts followed a predetermined strategy. Specifically, several rules were introduced in the competition to balance its results. The most important rules specific to the CIG’12 version of the competition are: 1) each maze is played for 4000 time units, after which the game progresses to the next level; 2) the game ends after a total of 24 000 time units; and 3) for each life remaining at the end of the game, Pac-Man is rewarded 800 points.

## III. MONTE CARLO TREE SEARCH

MCTS is a best-first search method based on random sampling of the state space for a specified domain [1], [2]. In gameplay, this means that decisions are made based on the results of random playouts. MCTS has been successfully applied to various turn-based games such as *Go* [25] and *Hex* [26]. Moreover, MCTS has been used for agents playing real-time games such as the *Physical Traveling Salesman* [5], real-time strategy games [4], and *Ms Pac-Man* [15]. A particular challenge for agents playing real-time games is that they are characterized by uncertainty, a large state space, and open-endedness. However, MCTS copes well when limited time is available between moves as it can be terminated anytime to select a move. Furthermore, the algorithm encapsulates uncertainty in its randomized playouts [3].

In MCTS, a tree is built incrementally over time and maintains statistics at each node corresponding to the rewards collected at those nodes and the number of times the nodes have been visited. The root of this tree corresponds to the agent’s current position. The basic version of MCTS consists of four steps, which are performed iteratively until a computational threshold is reached, i.e., a set number of iterations, an upper limit on memory usage, or a time constraint. The four steps (depicted in Fig. 1) at each iteration are as follows [24].

- **Selection.** Starting at the root node, children are chosen according to a selection policy. When a leaf node is reached that does not represent a terminal state, it is selected for expansion.
- **Expansion.** All children are added to the selected leaf node given available moves.
- **Playout.** A simulated playout is run, starting from the state of the added node. Moves are performed randomly

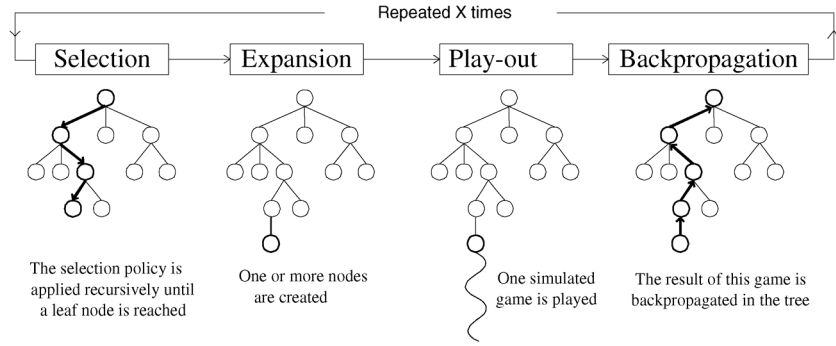


Fig. 1. Strategic steps of MCTS [24].

or according to a heuristic strategy until a terminal state is reached.

- **Backpropagation.** The result of the simulated playout is propagated immediately from the selected node back up to the root node. Statistics are updated along the tree for each node selected during the selection step and visit counts are increased.

A combination of moves selected during the selection and playout steps form a single simulation. During the selection step, moves are executed according to the selection policy, and during playout moves are performed (semi)randomly, or according to some simulation strategy. Because results are immediately backpropagated, MCTS can be terminated anytime to determine the decision to be made. Moreover, no static heuristic evaluation is required. However, in most cases, it is beneficial to add domain knowledge for choosing moves made during the playout.

#### IV. MCTS FOR *Ms Pac-Man*

This section discusses the enhancements to the MCTS framework for the Pac-Man agent. The agent builds upon the ideas proposed by Ikehata and Ito [15]. First, a description of the search tree is provided. In subsequent sections, the reward structure and individual enhancements to the strategic steps of MCTS are given. The goal is to present techniques that are both specific to *Ms Pac-Man*, but could also be extended or adapted to other (real-time) domains.

##### A. Search Tree and Variable Depth

The game's environment consists of four different mazes. These can be directly represented as a graph where the junctions are nodes, and corridors between junctions are edges. Pac-Man has the option to make a decision at any location in the graph. At a node she has a choice between more than two available directions, while on an edge she can choose to maintain her course or reverse. An example of such a graph is depicted in Fig. 2. The associated search tree is shown in Fig. 3.

Decisions in the tree are the moves made at nodes, i.e., junctions in the maze. Traversing the tree during the selection step means that Pac-Man moves along an edge until a node is reached. At this point, either a leaf is reached and playout starts, or a new edge is chosen based on a child of the current node. The tree represents a distinct discretization of *Ms Pac-Man*'s complex overall game state. As the tree is single player, the movements of the ghosts are not represented by nodes [15].

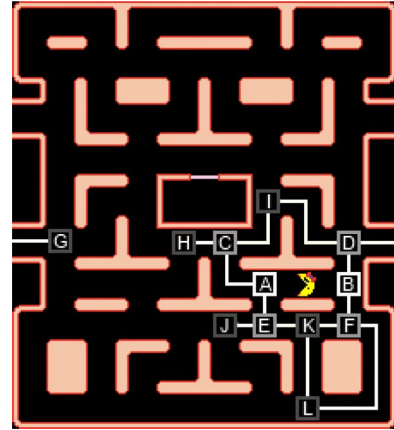


Fig. 2. Graph representation of a game state. All nodes refer to decisions for Pac-Man.

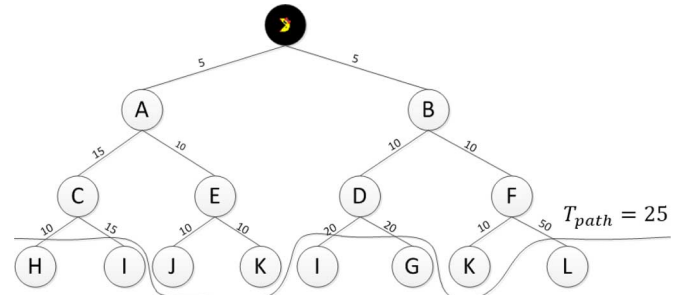


Fig. 3. Example tree with a distance limit of 25. Based on the game state in Fig. 2.

Instead, they are simulated based on a simulation strategy, while Pac-Man traverses the tree. As such, even within the tree, game states are approximations. Contrasting the approach used in traditional games, both the player and opponent(s) have decision nodes in the tree, and the game state represented by a node models the game state exactly for both players.

Within the tree, reverse moves, i.e., moves that lead back to a parent, are not considered beyond the first ply. When a node  $n_p$  representing junction  $j_p$  is expanded, each child  $n_i$  represents a move that leads to a different junction  $j_i$  in the maze, excluding junction  $j_p$ . Reverse moves are not considered deeper in the tree to ensure a larger difference in rewards between available moves. If reverse moves are expanded on every ply, more simulations per move are required in order to have conclusive differences in rewards. We consider the following: 1) in total, more nodes are expanded; and 2) the tree contains duplicate paths with similar or equal rewards.



Each node stores three different reward values,  $tactic \in \{\text{ghost}, \text{pill}, \text{survival}\}$ , both cumulative and maximized over their children's values. Suppose that node  $p$ , with a set of children  $C(p)$  has been visited  $N$  times. Furthermore, simulations made through  $p$  resulted in rewards  $R_{tactic,n}^p$  where  $n \in \{1, 2, \dots, N\}$ . The cumulative sum of rewards stored at  $p$  is

$$S_{tactic}^p = \sum_{n=1}^N R_{tactic,n}^p \quad (1)$$

where the rewards of simulation  $n$  through node  $p$ , i.e.,  $R_{ghost,n}^p$ ,  $R_{pill,n}^p$ , and  $R_{survival,n}^p$ , respectively, are defined in more detail in Section IV-E. The mean reward is then defined as

$$\bar{S}_{tactic}^p = \frac{S_{tactic}^p}{N}$$

The maximum mean reward is defined recursively as

$$M_{tactic}^p = \begin{cases} \bar{S}_{tactic}^p, & \text{if } p \text{ is a leaf} \\ -\infty, & \text{if } p \text{ is not in the tree} \\ \max_{i \in C(p)} M_{tactic}^i, & \text{otherwise.} \end{cases} \quad (2)$$

Reward quantities  $M_{tactic}$  and  $\bar{S}_{tactic}$  are both in  $[0, 1]$ .  $M_{tactic}$  is used when determining the value  $v_i$  for each move during selection, backpropagation, and deciding the move to make based on the current tactic. The different tactics are discussed in Section IV-B.

Ikehata and Ito [15] used a search tree restricted in depth to a fixed number of edges, without regard for the distance these edges represent in-game. Although the search tree in this paper is constructed similarly, the search path is variably determined by a distance limit  $T_{path}$ . A leaf is only expanded if the length of the path to the root node does not exceed  $T_{path}$  (as shown in Fig. 3). In *Ms Pac-Man*, distance units are represented using some internal measure, i.e., the distance between two consecutive pills is 4. The variable-depth search prevents the agent from choosing "quick fixes" when in danger, i.e., it may be safer to traverse a short path in the game when Pac-Man is in danger, than a long path which could be the case when tree depth is limited by a fixed number of edges. Furthermore, the scoring potential over all possible paths in the tree is normalized due to the uniform length of each path.

## B. Tactics

According to the current game state, a tactic [15] for determining the behavior of Pac-Man is selected. Tactics are based on the three subgoals of Pac-Man, and are dependent on the safety of the current position. Which tactic to choose is determined by setting a constant threshold  $T_{survival}$ , based on the highest survival rate of current simulations. At any time, one of the following is active [15].

- The ghost score tactic is selected if a power pill was eaten in a previous turn, edible ghosts are in the range of Pac-Man, and the maximum survival rate is above the threshold  $T_{survival}$ .
- The pill score tactic is the default tactic. It is applied when there are no edible ghosts in range, and the maximum survival rate is above the threshold  $T_{survival}$ .

- The survival tactic is used when the maximum survival rate of the previous or current search is below the threshold  $T_{survival}$ .

Survival heuristics have been used previously in sudden-death games such as *Connect6* [27]. However, unlike previous work, here we consider different tactics for the purposes of reward. Moreover, the current tactic is determined before MCTS starts and reassessed at every search.

The  $v_i$  value used for selection, backpropagation, and the final move selection is based on the current tactic. It is either the maximum survival rate when the survival tactic is active, or the current score multiplied by the survival rate for the ghost and pill tactics, respectively.  $v_i$  is defined as follows:

$$v_i = \begin{cases} M_{ghost}^i \times M_{survival}^i, & \text{if tactic is ghost} \\ M_{pill}^i \times M_{survival}^i, & \text{if tactic is pill} \\ M_{survival}^i, & \text{if tactic is survival.} \end{cases} \quad (3)$$

For the ghost and pill tactics, the maximum survival rate  $M_{survival}^i$  is used as a predictive indicator that the node's reward will be achieved in the long term.

## C. Search Tree Reuse

In real-time environments where fast decision making is required, MCTS may be at a disadvantage compared to strong rule-based algorithms. The quality of Monte Carlo sampling methods depends on the number of samples, i.e., more samples lead to better approximations, which allows for stronger play. However, the real-time 40-ms time frame in *Ms Pac-Man* severely limits the number of simulations possible.

In *Ms Pac-Man*, long-term reasoning is done using a preference relation over goals. That is, when Pac-Man makes a plan  $P_k$  to follow a path, out of the possible plans  $P_1, \dots, P_i$ , there exists a preference relation at the current turn such that  $P_k \succ \{P_1, \dots, P_i\} \setminus P_k$ . In subsequent turns, Pac-Man either confirms or denies the relevance of the current preference. Reasons to deny it in a later turn could be that Pac-Man would be in danger if she kept following the current plan, or there is an opportunity to score many points on an alternative path. However, if such a reason does not exist, the preference should be maintained until it no longer holds.

Rebuilding a new search tree at each turn restricts long-term planning. Since the information collected at each previous search is lost, the agent can become fickle and constantly switch back and forth between high-level preferences. However, reusing the search tree and its values each turn without modification can lead the agent to become too stubborn. Values collected from previous turns may become outdated, and consequently many new results may be required to overcome biases formed from past searches.

We combine two techniques for reusing the search tree between successive searches. The first technique uses a rule base to determine when the game state has significantly changed, indicating that the values stored in the tree are no longer relevant. The second technique decays the values stored in the tree between successive turns. This decay ensures that sample results from previous turns still play a role in making new decisions, but the importance of past results lowers over time. This decay is important because values collected in the past

were based on a situation that may have changed. However, in real-time domains, the state changes between successive turns are often small and so the information collected in previous searches often remains relevant.

1) **Rule-Based Reuse**: When reusing the search tree and its values, the game state can change in a manner not predicted by the search. In this case, values from previous searches should be discarded to make place for a new tree that represents the current game state. These cases were identified and constructed into a rule base. If any of the following is true, the existing tree is discarded, and a new tree is built.

- Pac-Man has died, or entered a new maze.
- Pac-Man ate a blue ghost or a power pill. This ensures that the search can immediately focus on eating (the remaining) ghosts.
- There was a global reverse. If all ghosts are forced to reverse, values of the previously built search tree no longer reflect the game state.

2) **Continuous Decay**: Values stored at nodes can be discounted to decay them over time, implying the continuous use of results throughout consecutive moves. The decay is achieved by multiplying each node's cumulative scores  $S_{ghost}$ ,  $S_{pill}$ , and  $S_{survival}$ , and visit count by  $\gamma$  at the start of each turn. Next, the search is performed with each node in the tree initialized with these discounted values. During the selection step and final move selection, the value of a node is determined by combining the decayed values, and the rewards of simulations made during the current move.

Reusing the tree implies that at the start of each move either a new root is selected based on Pac-Man's last move, or a new tree is built. If Pac-Man's last location was a junction, the new root is the first-ply child that corresponds to the last move selected. Next, the tree is restructured to make sure no reverse moves exist beyond the first ply. Moreover, the lengths of the paths in the tree are updated such that they correspond to the actual distances in the game. Last, the values stored at nodes that were altered are updated such that they equal the sum of their children.

The decaying of information we propose differs from previous applications of discounting used in MCTS. Discounted upper confidence bounds (UCB) was described on the multi-armed bandit problem in [28] and [29]. Discounted upper confidence bound applied to trees (UCT) is the application of discounted UCB to the tree case. In discounted UCT, discounting is applied every time a node is visited during the search. The rationale in this scheme is that earlier searches are biased by less informed choices since subtrees below each move have not been sufficiently explored, and so discounting is proposed to remove some of this bias. Discounting in this way did not appear to improve performance in *Othello*, *Havannah*, and *Go* [30]. When using a different MCTS algorithm [best recommendation with uniform exploration (BRUE)], this form of discounting has been shown to work better in practice in some domains and leads to a different bound on simple regret [31]. Accelerated UCT also discounts the node's values during the search but does so nonuniformly by taking into account each move's relative visit frequency [30].

In our case, a single discount is applied to all nodes in the tree between successive searches. The idea is to gradually de-

crease the importance of rewards collected over states further from the root, similar to how future rewards are discounted in Markov decision processes [32] and reinforcement learning [33]. This form of discounting during tree reuse may be appropriate in *Ms Pac-Man* since the brief search time leads to less accurate approximations, and the nodes in the tree do not perfectly model the actual state in the game due to abstraction assumptions. Searches performed later will in any case have a more up-to-date, and therefore more accurate, model of the actual game state.

Previous work used discount factors to decay heuristic values maintained to guide simulation strategies [34]. The results suggest that decaying these values ( $0 < \gamma < 1$ ) can help in the simulation strategies compared to no decay ( $\gamma = 1$ ) and no reuse ( $\gamma = 0$ ).

#### D. Selection and Expansion

During the selection step, a policy is required to explore the tree for rewarding decisions, finally converging to the most rewarding one. The upper confidence bound applied to trees (UCT) [1] is derived from the UCB1 policy [35] for maximizing the rewards of a multiarmed bandit. The UCT balances the exploitation of rewarding nodes while allowing exploration of lesser visited nodes. The policy that determines which child to select given the current node is the one that maximizes the following equation:

$$X_i = v_i + C \sqrt{\frac{\ln n_p}{n_i}} \quad (4)$$

where  $v_i$  is the score of the current child based on the active tactic, defined in (3). In the second term,  $n_p$  is the visit count of the node and  $n_i$  is the visit count of the current child.  $C$  is the exploration constant to be determined by experimentation.

The UCT selection policy is used when the visit count of all children is above a threshold  $T$  [2]. When one or more of the children's visit counts are below this threshold, a random uniform selection is made. In the case of *Ms Pac-Man*, the threshold used is 15, which ensures a high confidence on the safety of the path. An otherwise safe and/or rewarding node may have resulted in a loss the first time it was expanded, due to the stochastic nature of the game. Using the threshold ensures that this node will be explored again, increasing the confidence on the safety and reward of the decision.

#### E. Simulation

not a good practice

During simulation (the combined selection and playout steps), Pac-Man and the ghosts simultaneously make moves in a fully functional game state. Pac-Man's moves are determined by either the fixed selection performed by UCT, or a simulation strategy described in Section IV-F. Therefore, simulation consists of two parts: 1) the selection step; and 2) the playout step. Ghosts move according to their simulation strategy during both these steps, and have no fixed strategy determined by selection.

During the selection step, moves corresponding to each visited node during selection (Section IV-D) are performed by Pac-Man. This provides the basis for determining the achievable score of a path in the tree, while allowing for Pac-Man to be captured by the simulated ghost team. Whenever selection ends or

is interrupted, ployout begins and both Pac-Man and the ghosts move according to the simulation strategy.

There exist two cases in which the selection step can be interrupted due to an unpredictable change in the game state.

- 1) If Pac-Man loses a life during the selection step, a ployout is started from the last visited junction. Losing a life during the selection step is basically a suicide move, as Pac-Man may only travel forward without regard for safety. Therefore, the ployout is run starting from the last junction, to determine whether Pac-Man could have avoided the loss of life.
- 2) Pac-Man eats a ghost or a power pill; in this case, ployout is started immediately. However, if there are edible ghosts active and a power pill is eaten, simulation terminates immediately. This prevents Pac-Man from "wasting" power pills.

In both cases, the result of the simulation is backpropagated from the last visited junction, i.e., node in the search tree. Note that, in this case, it is not the leaf visited during selection that gets updated, rather backpropagation is started at an internal node. However, if the selection step is not interrupted, ployout starts from the junction represented by the leaf.

A game of *Ms Pac-Man* ends when either Pac-Man loses all lives, or after 24 000 time units. It is neither useful nor computationally achievable within the strict time limit of 40 ms to run numerous simulations until one of these conditions holds. Running long ployouts introduces bias for MCTS in favor of the heuristics used. Moreover, it updates the tree with results based on long-term estimations on the game state, which are neither accurate nor interesting for evaluating the current state.

The goal of the ployouts is to determine the short- and long-term safety and reward of a selected path. Therefore, different stopping conditions should be used. Two natural stopping conditions can be considered, either Pac-Man loses a life (dies), or the game progresses to the next maze. However, to prevent lengthy ployouts, additional stopping conditions are introduced. Therefore, during ployout, moves are made until one of the following four conditions applies.

- 1) A preset number of time units  $T_{\text{time}}$  have passed [15].
- 2) Pac-Man is considered dead, i.e.: a) came into contact with a nonedible ghost; and b) is trapped at the end of the ployout, and every available path is blocked by ghosts.
- 3) The next maze is reached.

$T_{\text{time}}$  is determined by adding the distance limit of the search tree  $T_{\text{path}}$  with a simulation time constraint  $T_{\text{simulation}}$ . This ensures that each simulation has the same length in time and, in effect, the same scoring potential.

When a simulation ends for any of the aforementioned reasons, Pac-Man's score is determined based on the three subgoals of the game. Results of a simulation consist of three values [15]:

- 1)  $R_{\text{survival}} = \begin{cases} 0, & \text{if Pac-Man died} \\ 1, & \text{if Pac-Man survived;} \end{cases}$
- 2)  $R_{\text{pill}} \in [0, 1]$  is the number of pills eaten, normalized by the number of pills at the start of the level;
- 3)  $R_{\text{ghost}} \in [0, 1]$  is the number of ghosts eaten, normalized by the total edible time at the start of the simulation.

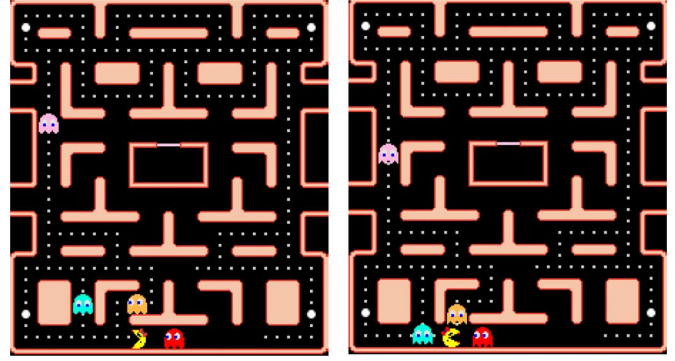


Fig. 4. Ghosts executing a pincer move.

not a good practice

These rewards are defined as described in Section IV-G. Each node stores a tuple containing the sum of these individual rewards  $S_{\text{ghost}}$ ,  $S_{\text{pill}}$ , and  $S_{\text{survival}}$ , as defined in Section IV-C.

The goal for Pac-Man during the ployout is acquiring the highest score possible while avoiding a loss of life. The ghosts have three goals: 1) ensure that Pac-Man loses a life by trapping her, i.e., every possible path leads to a nonedible ghost; 2) ensure the lowest ghost reward  $R_{\text{ghost}}$ , which increases when Pac-Man eats edible ghosts; and 3) limit as much as possible the number of pills Pac-Man can eat. To achieve these goals, we introduce a simulation strategy for Pac-Man and the ghosts in Section IV-F.

#### F. Simulation Strategy

repeat sentence as in IV-E?

During simulation, Pac-Man and the ghost team's moves are executed simultaneously in a fully functional game state, using the complete rule set of the game (Section II). In this section, the simulation strategies for the ghosts and Pac-Man are detailed. The strategies were designed to ensure that any time Pac-Man does not survive the ployout (Section IV-E); it is due to all possible paths being blocked by ghosts. Therefore,  $S_{\text{survival}}$  can be considered as an indicator of the probability of a pincer move occurring along the path.

1) **Ghost Simulation Strategy:** The goal of the ghosts is to trap Pac-Man in such a way that every possible move leads to a path blocked by a ghost, i.e., a pincer move (Fig. 4). The ghosts, therefore, are assigned a random target-location vector target that determines whether an individual ghost is to approach the front or rear of Pac-Man. This approach is similar to [15], with the addition of using a randomized target vector.

Ghosts move based on an  $\epsilon$ -greedy strategy [33], [36]. With probability  $\epsilon = 0.2$  at each turn, a ghost makes a random move. With probability  $1 - \epsilon$ , the ghosts move according to strategic rules, derived from the rules proposed in [15] and detailed in this section. When selecting a ghost's move during simulation, there are two mutually exclusive cases to consider: nonedible or edible. Moreover, there is a third case, which overrules a selected move in any case.

**Case 1:** If ghost  $g_i$  is not edible, a move is selected according to the following rules.

- 1) The ghost selects a move that immediately traps Pac-Man if available.
- 2) If the ghost is in the immediate vicinity of Pac-Man, i.e., within six distance units, the ghost moves along the next direction of the shortest path to Pac-Man.



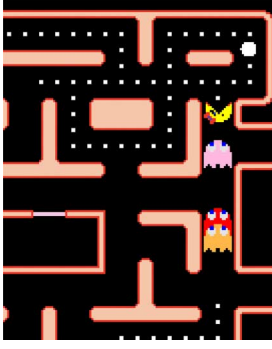


Fig. 5. Ghosts chasing Pac-Man from similar distance and location.

- 3) If the ghost is closer to the junction that lies in front of Pac-Man than Pac-Man is herself, and  $\text{target}_i = \text{front}$ , then the ghost will move directly to Pac-Man's forward junction, which essentially blocks her forward path.
- 4) If the ghost is on a junction directly connected to the edge that Pac-Man is located on and if no other ghost is on the edge moving in the same direction, then the ghost chooses the move that leads to this edge, blocking Pac-Man based on the  $\text{target}_i$  designation.
- 5) Otherwise, the ghost moves closer to the assigned target location. Based on the value of  $\text{target}_i$ , the target is either the nearest junction in front or behind Pac-Man.

**Case 2:** If ghost  $g_i$  is edible, a move is chosen that maximizes the distance between him and Pac-Man.

**Case 3:** If a ghost is to move on an edge currently occupied by another ghost moving in the same direction, the move is eliminated from the ghost's selection and a different move is selected randomly. This policy ensures that ghosts are spread out through the maze, increasing their possible trapping or catching locations. It also prevents multiple ghosts from chasing Pac-Man at the same (or similar) distance and location shown in Fig. 5.

**2) Pac-Man Simulation Strategy:** Moves made by Pac-Man are prioritized based on safety and possible reward. When more than one move has the highest priority, a move is chosen at random. Before discussing the strategy in detail, the concept of a safe move must first be defined. A safe move is any move that leads to an edge which:

- has no nonedible ghost on it moving in Pac-Man's direction;
- whose next junction is safe, i.e., in any case, Pac-Man will reach the next junction before a nonedible ghost.

During playout, Pac-Man moves according to the following set of rules. If Pac-Man is at a junction, the following rules apply, sorted by priority.

- 1) If there are reachable edible ghosts, i.e., the traveling time it takes to reach a ghost is smaller than its edible time remaining, then select a safe move that follows along the shortest path to the nearest edible ghost.
- 2) If a safe move leads to an edge that is not cleared, i.e., contains any number of pills, it is performed.
- 3) If all safe edges are cleared, select a random move leading to a safe edge.
- 4) If no safe moves are available, a random move is selected [15].

If Pac-Man is on an edge, she can either choose to maintain her current course or reverse it. The following rules consider the cases when Pac-Man is allowed to reverse:

- there is a nonedible ghost heading for Pac-Man on the current path;
- a power pill was eaten; in this case, the move that leads to the closest edible ghost is selected.

In any other case, Pac-Man continues forward along the current edge. Note that if Pac-Man previously chose to reverse on the current edge, she may not reverse again until she reaches a junction. Moreover, to improve the performance of playouts, checking the first condition is only performed if the last move made at a junction was based on an unsafe decision.

### G. Long-Term Goals

Time is an important aspect of *Ms Pac-Man*. Ghosts need to be eaten as fast as possible, and remaining in a maze longer than necessary increases the risk of being captured and decreases the total time available for scoring points. Furthermore, after 10 000 points, Pac-Man gains a life. These are examples of long-term goals in the game. Any MCTS implementation looks at short-term rewards when running simulations. To estimate the rewards of these long-term goals, specific results are considered for both pill and ghost rewards.

To encode the long-term goal in the ghost reward  $R_{\text{ghost}}$ , its immediate reward when eating a ghost is the edible time remaining before the ghost was eaten. This ensures there is a preference for eating ghosts as fast as possible. Furthermore, when Pac-Man eats a power pill (while no edible ghosts are active) during simulation, she must achieve a ghost score higher than 0.5 at the end of the playout. If this is not the case, i.e., the ghosts were too far away to be eaten in time, the pill-reward  $R_{\text{pill}}$  is 0. If the minimum ghost score of 0.5 is achieved after eating a power pill, the pill reward is increased:  $R_{\text{pill}} = R_{\text{pill}} + R_{\text{ghost}}$ . This rapid change in scores enables Pac-Man to wait for the ghosts to be close enough before eating a power pill.

There are only four power pills available per maze to provide a guaranteed safe escape. In addition, clearing a maze fast increases the scoring potential as the game ends after 24 000 time units. Therefore, eating all pills before the game naturally progresses to the next maze is a beneficial long-term goal. To shape the reward  $R_{\text{pill}}$  accordingly, we introduce an edge reward. This means that the reward for eating pills is incremented only when an edge is cleared, i.e., the last pill on the edge is eaten. This ensures that Pac-Man prefers to eat all pills on the edges she visits, rather than leaving isolated pills, which become hard to reach as the game progresses. On the one hand, long edges in the maze have a higher scoring potential; on the other hand, they pose a greater risk for Pac-Man. Therefore, it is generally beneficial to clear long edges when it is safe to do so. Although the edge reward resolves the issue partially, clearing several short edges may still result in a larger reward than clearing a single, long edge. To incorporate this nonlinear goal in scoring, the edge reward is defined as follows:  $R_{\text{edge}} = \text{num pills}(e_i)^p$ , where  $1 < p < 2$  and  $e_i$  represents the edge cleared. This ensures that long edges provide relatively higher rewards while restricting the maximum reward per edge. In this case,  $R_{\text{pill}} = \max(R_{\text{pill}}, R_{\text{edge}})$  and is normalized based on the total number

of pills in the maze. This ensures that the simulation is properly rewarded, even if no edges were fully cleared.

### H. Backpropagation and Move Selection

Results are backpropagated from either the expanded leaf node, or the internal node reached in the interrupted selection step, to the root based on maximum backpropagation [2]. Scores stored at each internal node represent the maximum scores of its children based on  $v_i$ , according to the current tactic, as defined in (3). Whereas traditionally, MCTS implementations for games use average backpropagation, maximization is applied to MCTS for *Ms Pac-Man*. During the search when a state is reached, upon return to the parent, the maximum score  $M_{\text{tactic}}$  over the children is returned to the parent. The maximum is used because every move Pac-Man can make at a junction can have altogether different results [15]. For example, at a given junction, Pac-Man has two options to move. A decision to go left leads to a loss of life for Pac-Man in all simulations, whereas a choice to go right is determined to be safe in all simulations. When using average values, the resulting score is 0.5, whereas maximum backpropagation results in the correct evaluation of 1.

After each simulation, results are incremented starting at the node from which the playout step was initiated. Next, the values of the parent node are set to the maximum values over all its children. This process is recursively executed until the root is reached, according to the definition of  $M_{\text{tactic}}$  in (2). Children of the root then obtain the scores of their highest scoring leaf, and the root has the value of the highest scoring node in the tree.

For final move selection, first the current tactic (Section IV-B) is evaluated. If the maximum survival rate is below the threshold, i.e.,  $M_{\text{survival}} < T_{\text{survival}}$ , selection falls back to the survival tactic and backpropagates values from all leaf nodes based on maximizing  $M_{\text{survival}}$ . Otherwise, scores are determined based on the current tactic. The scores of the nodes in the first ply are evaluated and the node with the highest  $v_i$ , as defined in (3), which has  $M_{\text{survival}} > T_{\text{survival}}$ , is submitted. If the current tactic provides no feasible reward, i.e.,  $v_i = 0$  for every node, it is replaced according to the ordering ghost, pill, survival. This occurs when, e.g., the nearest pill or edible ghost is out of the search tree's range.

### I. Pseudocode

The algorithm for a single simulation of the MCTS implementation is summarized in Algorithm 1.

**Algorithm 1:** Pseudocode of MCTS for *Ms Pac-Man*.

```


1 MCTS(node  $p$ , cumulative path length  $l$ ):
2   if  $l > T_{\text{path}}$  then
3     return Playout( $p$ )
4   else if ExpandableNode( $p, l$ ) then
5     for  $c \in C(p)$  do
6        $c \leftarrow \text{NewLeafNode}()$ 
7       Add new leaf  $c$  to the tree
8       ( $R, c'$ )  $\leftarrow$  Playout( $p$ )
9       Update( $c, R$ ); Update( $p, R$ )

```

```

10  return ( $p, R$ )
11 else
12   Let  $t$  be the tactic set at the root
13    $n_p \leftarrow p.n_{\text{old}} + p.n_{\text{new}}$ 
14   for  $c \in C(p)$  do
15     Let  $i$  be the action associated with child  $c$ 
16      $n_i \leftarrow c.n_{\text{old}} + c.n_{\text{new}}$ 
17      $v_i \leftarrow M_t^i$  as defined in (3)
18   Select a move  $i$  that maximizes  $X_i$  from Eq. (4)
19   ( $c, R', \Delta l$ )  $\leftarrow p.$  ApplyMove( $i$ )
20   if  $R'_{\text{survival}} = 0$  then
21     ( $R, c$ )  $\leftarrow$  Playout( $p$ )
22     Update( $p, R$ )
23   return ( $p, R$ )
24    $R \leftarrow \text{MCTS}(c, l + \Delta l)$ 
25   Update( $p, R$ )
26   return ( $p, R$ )

```

Starting from the root, each recursive call passes down the new child state until an expandable node is reached, then its children are added to the tree and a playout is started from one of these new children. Each node stores several scores, including the one used to guide the search depending on the current tactic, as described in Section IV-B. Also, two versions of each average score  $\bar{S}_t$  are maintained, both the old and new versions. The old scores ( $S_{t,\text{old}}$ ) represent the sum of decayed values from past searches, and the new scores ( $S_{t,\text{new}}$ ) are maintained for the current search. These values are updated between searches based on (5) and (6) .

As described in Section IV-A, the depth of the tree is variable depending on the in-game distance between nodes. So, the search maintains a cumulative distance from the root  $l$  in order to check whether the search has passed beyond its distance limit  $T_{\text{path}}$ , on line 2. A node is expandable if its distance from the root is below the distance limit  $T_{\text{path}}$  and none of its children are in the tree. Nodes are expanded by adding all the children to the tree as seen between lines 4 and 9. The **PLAYOUT** procedure simulates the rest of the game as described in Section IV-F, and returns a tuple  $R = (R_{\text{ghost}}, R_{\text{pill}}, R_{\text{survival}})$  paired with the child state  $c$  that was first encountered from  $p$ .

The **UPDATE** procedure at node  $p$  adds to the appropriate cumulative score based on the current tactic  $S_{t,\text{new}}^p$ , increments the survival score  $S_{\text{survival,new}}^p$ , and increments the number of visits at the specified node  $n_{\text{new}}$ . In addition, the maximum value  $M_t$  is updated based on (2). Note that the decay between searches does not change the value of  $M_t$  since the decay equations (5) and (6) always preserve the average values  $\bar{S}_t$ .

The next state is obtained by the simulation from node  $p$  in line 19. The MCTS search space is discretized as described in Section IV-A, so the transition from **APPLYMOVE** represents the simulation between the current node  $p$  to child  $c$ . If *Ms Pac-Man* dies during the transition in the MCTS tree, then instead of continuing with the current transition, **PLAYOUT** is invoked from the parent. As seen between lines 20 and 22, a playout is started from  $p$  onward. Otherwise the length of the path along  $p \rightarrow c$  returns as  $\Delta l$  and the search continues from  $c$  on line 24.



TABLE I  
BASELINE SCORES, 400 GAMES

Pac-Man agent: MCTS PAC-MAN				
Ghost Agent	Score		Lives Avg.	Maze Avg.
	Avg.	95% c.i.		
LEGACY2	82,689	1.06%	2.49	7.66
FLAMEDRAGON	54,287	1.07%	2.75	7.82
WILSH	30,839	4.42%	0.04	3.39
GHOSTBUSTER	6,508	3.57%	0.00	0.04
MEMETIX	5,686	3.64%	0.00	0.03

Between successive searches, two important updates are made to the tree. First, the tree is restructured as described in Section IV-C. Then, the following updates are applied at each node of the new tree and each tactic  $t$

$$S_{t,old} \leftarrow \gamma(S_{t,old} + S_{t,new}), \quad S_{t,new} \leftarrow 0 \quad (5)$$

$$n_{old} \leftarrow \gamma(n_{old} + n_{new}), \quad n_{new} \leftarrow 0. \quad (6)$$

## V. EXPERIMENTS

In this section, the experimental setup is described and detailed, and experimental results discussed. Moreover, in Section V-C, results of the 2012 World Congress on Computational Intelligence (WCCI'12, Brisbane, Qld., Australia) and CIG'12 competitions are presented.

### A. Experimental Setup

The MCTS Pac-Man agent was implemented using the framework provided by the *Ms Pac-Man Versus Ghosts Competition* [8]. The *LEGACY2THERECKONING* (*LEGACY2*) ghost team, provided by the framework provides a baseline. Moreover, several competing ghost teams have released the source code for their agent, allowing us to run experiments using different ghost teams that have entered the CIG'12 competition.

The version of the *Ms Pac-Man* game framework used is CIG12\_6.2. It includes precomputed distances for the four mazes, providing a fast method for determining shortest paths and distances. Furthermore, it asserts the new rules introduced in the CIG'12 edition of the competition. Note that, due to this change, results from previous competitions cannot be directly compared to those presented in this paper.

For each experiment, 400 games are played, allowing the agents the official 40 ms to compute a move each turn. Moreover, as the competition rules state, it is not allowed to use more than one thread for computation, therefore multithreaded operations are not considered in any of the experiments. Games played versus the *LEGACY2* ghost team generally last the complete 24 000 time units. Thus, every game lasts approximately 16 min. Against the other ghost teams games rarely progress to the second maze, lasting on average 3000 time units, i.e., 2 min. Consequently, running 400 experiments takes 4.4 days on a single AMD64 Opteron 2.4-GHz processor.

The following values, manually tuned, were used for the parameters discussed in this paper: the minimum survival rate  $T_{survival} = 0.6$ , the decay factor for tree reuse  $\gamma = 0.6$ , the distance limit  $T_{path} = 40$ , the maximum time units per payout

$T_{simulation} = 60$ , and UCT constant  $C = 1$ . These values were optimized against the *LEGACY2* ghost team. Note that, against different ghost teams, other parameter values may result in improved results. Parameters that fall within the  $[0, 1]$  range were tuned with 0.1 intervals, path and simulation length with intervals of ten units. All parameters were tuned independently. However, each time a set of parameters was found to improve results, these parameters were combined in order to determine their combined performance. Tree reuse was enabled for both  $S_{pill}$  and  $S_{survival}$ ; ghost scores stored at nodes were not reused between turns. Ghost scores are significantly different than the other scores, and may require an altogether different set of parameters.

To determine the influence of the proposed enhancements, results are compared to agents with a single enhancement disabled. Additional experiments are run using agents with: 1) a fixed edge depth limit; 2) a randomized simulation strategy; 3) a new tree for every search; 4) reuse of the tree with no decay; and 5) no long-term goals enabled.

### B. Results

Experiments were run to evaluate the performance of the MCTS Pac-Man agent against the benchmarking ghost team *LEGACY2* provided by the competition framework. Beside this agent, the performance of our agent was tested versus four ghost teams that released their source code publicly. They are *MEMETIX*, *GHOSTBUSTER*, *FLAMEDRAGON*, and *WILSH*; ranked second, third, sixth, and eighth, respectively, in the CIG'12 competition. The highest ranked agent *MEMETIX* performs a depth-2 minimax search combined with a heuristic evaluation function, whereas *GHOSTBUSTER*, *FLAMEDRAGON*, *LEGACY2*, and *WILSH* use a specific rule-base for making decisions. Results consist of the average score, the average number of lives remaining, and the average maze reached at the end of each game. Average scores are rounded to the nearest integer. When comparing results, the "Score Decrease" column refers to the relative change in score compared to the baseline results in Table I.

In Table I, the baseline scores are presented. These are the results of 400 games played against each ghost team. All enhancements discussed are enabled for the agent. For every table in this paper, the values listed in the " $\pm 95\%$  c.i." column represent a score range that corresponds to a 95% confidence interval with respect to the average score. For instance, a 95% confidence interval for the score against *LEGACY2* is  $82\,689 \pm 876.50$  (1.06%).

According to the game's rules, each level ends after 4000 time units. Based on the total time limit of 24 000 time units, if Pac-Man survives, the sixth maze can be reached. The results show that, versus the *LEGACY2* ghost team, our agent on average reaches 7.66 mazes. This would imply that it takes 3133 time units per maze. Moreover, on average, it has 2.49 lives remaining at the end of each game. There is certainly a tradeoff between safe play and achieving high scores. When a Pac-Man agent plays too cautiously, it will miss opportunities to score points, whereas, when playing with a lower safety threshold  $T_{survival}$ , the risk of losing lives faster against strong opponents restrains scoring potential.

To investigate the influence of individual enhancements, games were played using Pac-Man agents with individual enhancements disabled or defaulted.

- 1) A constant tree-depth of five edges, determined by trial and error, replaces the variable-depth tree enhancement. The maximum depth of the tree is limited by five edges, instead of variably limited using the distance limit discussed in Section IV-A.
- 2) The simulation strategy discussed in Section IV-F was replaced by a random strategy for Pac-Man and an  $\epsilon$ -greedy strategy for the ghost team. In particular, Pac-Man cannot reverse and chooses each move randomly, and the ghosts choose the path that leads closer to Pac-Man with probability 0.95 and with uniform probability 0.05.
- 3) Tree reuse, as described in Section IV-C, is completely disabled, forcing the agent to build a new tree each turn.
- 4) Tree reuse is enabled with decay factor  $\gamma = 1$  (no decay). The tree is still discarded when the rules described in Section IV-C apply.
- 5) The long-term goals described in Section IV-G are disabled. The pill reward is defaulted as  $R_{\text{pill}} = \text{pills}_{\text{eaten}} / \text{pills}_{\text{total}}$ , the number of pills eaten, normalized by the total number of pills in the maze. Moreover, the ghost reward does not take into account the time at which ghosts were eaten, i.e.,  $R_{\text{ghost}} = \text{ghosts}_{\text{eaten}} / 4$ .

Table II shows comparative results for individual enhancements. The simulation strategy has the largest impact on overall performance, followed by using a fixed depth tree and tree reuse with decay, respectively. While the number of simulations per move increases when employing a randomized simulation strategy, the quality of the simulations decreases significantly. Most importantly, it causes Pac-Man to die in simulations due to reasons other than pincer moves.

Overall, the enhancements have a different influence when playing against different ghost teams. This is most prevalent in case 4, where performance decrease differs as much as 20% between GHOSTBUSTER and LEGACY2. Similarly, there is a significant difference in the results for the other cases. The reason for this is twofold. First, as mentioned, all parameters were tuned against the LEGACY2 ghost team. Using different parameter settings against different ghosts may result in more consistent results. However, no parameter tuning was performed against the other ghost teams. Second, the strategies each ghost team use are considerably different. Thus, any technique that works well against one ghost team may fail against another. The goal is to find a strategy that does not hurt performance (significantly) against most ghost teams, but increases performance overall.

Both tree reuse and the use of decay between moves improve performance against all ghost teams. This shows that in *Ms Pac-Man*, results from previous searches can be successfully used during consecutive searches. The results show that information reuse improves scores against all tested ghost teams. Furthermore, using a decay factor dominates tree reuse without any decay (case 4). It makes sure that biases developed during previous searches are dismissed over time.

Comparing between the cases, long-term goals seem to help the least. In one instance, against FLAMEDRAGON, enabling long-term goals leads to worse performance. The long-term

TABLE II  
SINGLE ENHANCEMENTS DISABLED, 400 GAMES

Pac-Man agent: MCTS PAC-MAN					
Ghost Agent	Avg.	Score Decrease	95% c.i.	Lives Avg.	Maze Avg.
<b>1. Fixed depth</b>					
LEGACY2	79,770	3.53%	1.25%	2.55	7.63
FLAMEDRAGON	51,697	4.77%	1.04%	2.73	7.73
WILSH	30,246	1.92%	4.49%	0.05	3.37
GHOSTBUSTER	5,579	14.28%	4.18%	0.00	0.05
MEMETIX	4,583	19.39%	4.53%	0.00	0.04
<b>2. Random simulation</b>					
LEGACY2	33,465	59.53%	1.88%	1.63	6.47
FLAMEDRAGON	15,546	71.36%	4.36%	0.31	3.40
WILSH	5,289	82.85%	4.11%	0.00	0.45
GHOSTBUSTER	2,631	59.58%	2.98%	0.00	0.00
MEMETIX	2,294	59.66%	3.12%	0.00	0.00
<b>3. No reuse</b>					
LEGACY2	76,083	7.99%	1.13%	2.58	7.32
FLAMEDRAGON	49,332	9.13%	1.15%	2.90	7.38
WILSH	28,816	6.56%	4.35%	0.03	3.12
GHOSTBUSTER	5,280	18.88%	3.80%	0.00	0.02
MEMETIX	5,155	9.33%	3.70%	0.00	0.02
<b>4. No decay</b>					
LEGACY2	74,532	9.86%	1.23%	2.25	7.07
FLAMEDRAGON	48,891	9.94%	1.10%	2.43	7.27
WILSH	24,050	22.02%	4.46%	0.01	2.51
GHOSTBUSTER	4,565	29.86%	4.60%	0.00	0.01
MEMETIX	5,182	8.87%	3.84%	0.00	0.02
<b>5. No long-term goals</b>					
LEGACY2	80,615	2.51%	1.16%	2.55	7.78
FLAMEDRAGON	56,974	-4.95%	1.13%	2.67	7.92
WILSH	28,143	8.74%	4.32%	0.02	2.94
GHOSTBUSTER	5,912	9.17%	3.42%	0.00	0.03
MEMETIX	5,689	-0.06%	3.41%	0.00	0.02

goals make assumptions about the behavior of the ghosts, therefore incorrect assumptions can lead to problems when planning. Despite this result against FLAMEDRAGON and the lack of improvement against MEMETIX, three out of five cases show that long-term goals do improve the average score. Hence, long-term goals were enabled in the competition version of the agent.

In order to determine the performance of tree search in combination with Monte Carlo sampling, experiments without search beyond the first ply were run. In Table III, results are presented of these runs. Using 1) UCT selection as described in Section IV-D, and 2) a random uniform selection, where each node is selected with equal probability. Table IV contains results of the same experiments, without the use of the simulation strategy.

When using search, the agent becomes more aware of pincer moves that will probably not occur against weak opponents. Against weak ghost teams such as LEGACY2, our agent has close to perfect performance, i.e., clearing mazes as fast as possible, and eating all ghosts for each power pill. Moves that would be unsafe versus strong agents are safe against weaker ones, and as a result the agent plays too cautiously. However, the better the ghost team performs, the more search improves the results, as shown in Table III, provided that the simulation strategy is used. Moreover, as shown in Table IV, the simulation strategy skews the results. When we compare results 3 and 4 to the random payout presented in Table II, search improves results in almost every case.

TABLE III  
DEPTH-1 SEARCH, SIMULATION STRATEGY, 400 GAMES

Pac-Man agent: MCTS PAC-MAN					
Ghost Agent	Avg.	Score Decrease	95% c.i.	Lives Avg.	Maze Avg.
<b>1. UCT selection</b>					
LEGACY2	78,444	5.13%	1.21%	2.84	7.54
FLAMEDRAGON	51,160	5.76%	1.00%	2.91	7.55
WILSH	28,207	8.53%	4.75%	0.05	3.01
GHOSTBUSTER	5,354	17.73%	4.52%	0.00	0.04
MEMETIX	4,163	26.79%	4.74%	0.00	0.02
<b>2. Uniform random selection</b>					
LEGACY2	65,680	20.57%	0.85%	3.15	6.11
FLAMEDRAGON	43,874	19.18%	1.17%	2.93	6.38
WILSH	27,982	9.27%	3.83%	0.04	3.01
GHOSTBUSTER	5,490	15.64%	3.71%	0.00	0.01
MEMETIX	5,748	-1.09%	3.28%	0.00	0.01

TABLE IV  
DEPTH-1 SEARCH, RANDOM SIMULATION, 400 GAMES

Pac-Man agent: MCTS PAC-MAN					
Ghost Agent	Avg.	Score Decrease <sup>a</sup>	95% c.i.	Lives Avg.	Maze Avg.
<b>3. UCT selection / Random simulation</b>					
LEGACY2	22,390	33.09%	2.32%	1.93	5.37
FLAMEDRAGON	12,245	21.24%	4.22%	0.42	3.10
WILSH	4,904	84.10%	7.28%	0.00	0.51
GHOSTBUSTER	2,532	3.78%	2.99%	0.00	0.00
MEMETIX	2,392	-4.3%	2.62%	0.00	0.00
<b>4. Uniform random selection / Random simulation</b>					
LEGACY2	18,295	45.33%	2.28%	2.10	4.56
FLAMEDRAGON	10,610	31.75%	4.25%	0.43	2.82
WILSH	4,748	10.23%	3.65%	0.00	0.59
GHOSTBUSTER	2,362	10.23%	2.68%	0.00	0.00
MEMETIX	2,211	3.59%	2.42%	0.00	0.00

<sup>a</sup> Relative to Table II, case 2.TABLE V  
PAC-MAN SIMULATION STRATEGY, 400 GAMES

Pac-Man agent: Simulation strategy					
Ghost Agent	Avg.	Score Decrease	95% c.i.	Lives Avg.	Maze Avg.
<b>No search</b>					
LEGACY2	14,058	83.00%	5.00%	0.00	1.42
FLAMEDRAGON	17,168	68.38%	4.60%	0.20	2.89
WILSH	5,626	81.76%	4.31%	0.00	0.19
GHOSTBUSTER	3,052	53.11%	3.64%	0.00	0.00
MEMETIX	3,124	45.05%	3.68%	0.00	0.00

Finally, Table V presents the scores when both tree search and Monte Carlo sampling are disabled. Effectively, this means that the Pac-Man simulation strategy discussed in Section IV-F determines the move to make.

Based on the difference between these results and those from the experiments presented in Table III, we can conclude that **sampling the state space, even for a short 40 ms, is a valid approach for playing *Ms Pac-Man*.**

### C. Competition Results WCCI'12 and CIG'12

Our MCTS Pac-Man agent was entered in the *Ms Pac-Man* Versus Ghost Competition [8] held for the WCCI'12 and CIG'12 conferences under the nickname MAASTRICHT. The

TABLE VI  
PAC-MAN RANKINGS WCCI'12, TOP 15/63

Rank	Agent name	Avg. score
1	EIISOLVER	91,449.90
2	MAASTRICHT	87,431.00
3	ICEP-FEAT-SPOOKS	86,570.90
4	MEMETIX	74,158.80
5	GHOSTBUSTER	71,868.10
6	HAIMAT	63,649.80
7	SIR_MACELON	62,910.80
8	ARTHURSPONER	60,423.90
9	EGREAVETTE	57,036.50
10	RCPINTO	41,682.80
11	GREENTEA	34,885.10
12	ALHEJALI	26,558.70
13	FINDIG	26,409.00
14	XTILE	25,875.10
15	STEFAN	25,394.50

TABLE VII  
PAC-MAN RANKINGS, CIG'12, TOP 15/36

Rank	Agent name	Games	Ranking	RD
1	MAASTRICHT	91	2214.7	29.9
2	SIR_MACELON	91	2208.8	29.9
3	ICEP-IDDFS	87	2172.8	29.9
4	EIISOLVER	282	2030.9	25.9
5	GHOSTBUSTER	299	2010.6	26.8
6	SOFAKINGBOSS	300	1973.8	26.3
7	MEMETIX	295	1900.4	26.8
8	ICEP-MDFS	86	1884.2	29.8
9	LOCUTUS	77	1790.8	29.9
10	SPIDERFISH	181	1782.9	24.8
11	KING	76	1731.4	29.6
12	MONE	80	1729.9	29.9
13	SARDINHA	294	1687.9	24.8
14	NEUROTIC	74	1671.2	29.7
15	EVOLVED	78	1600.3	30.0

competition's results for the WCCI'12 edition are based on the average results of each Pac-Man agent in a **round-robin** tournament against every other ghost team. An older version of our agent was entered in this competition (cf., [23]). The results of the Pac-Man agents are presented in Table VI, showing a second place for our agent out of **63 contestants**.

For the CIG'12 competition, results were based on a Glicko-based pairing in which each agent has a rating based on the games won against other agents. Based on these ratings, agents are paired with appropriate opponents. A game is won when an agent has a higher score against a matched ghost team than the paired opponent versus the same ghost team. The agent's ranking is then increased based on the ranking of the opponent defeated. In this competition, our *Ms Pac-Man* agent achieved a first place of **36 competitors, winning 67.6% of the games** (Table VII).

Unfortunately, the methods used by other strong agents in the competitions are not currently known in detail. However, based on inspection of a publicly available source code, we determined that both EIISOLVER and MEMETIX use a minimax approach equipped with a heuristic evaluation function. The Pac-Man agent ICEP-FEAT-SPOOKS is an enhancement of the controller SPOOKS, winner of the CIG'11 competition, which uses a rule-based approach [37].



## VI. CONCLUSION

In this paper, an MCTS agent and enhancements were described and shown to be suitable and effective for *Ms Pac-Man*. The agent was shown to compete with strong opponents, and emerges in high ranking results. The enhancements introduced provide an overall improvement on performance, where the simulation strategy is responsible for the most considerable increase in score. Moreover, we have shown that using tree search ensures that the agent achieves high scores versus both strong and weak ghost teams. Furthermore, we have shown a range of enhancements that improved the performance of our agent against either all tested opponents or against most opponents. Most notably, an increase in performance was achieved when reusing information between moves and decaying this information with a factor  $\gamma$ . This approach differs considerably from previous work where either all information was reused as is, or decaying was performed during the search, as opposed to in between searches.

Whether these enhancements translate to similar results in different real-time domains remains an open research question. In particular, information reuse using a continuous decay may improve performance without adding specific domain knowledge. Modifying the rewards to encompass long-term goals will require knowledge about the domain to implement successfully, as will the simulation strategy. However, research has shown that fast reactive agents can be trained using genetic algorithms [9], [38], and implemented as a simulation strategy for MCTS [39].

When playing versus strong ghost teams such as the ones used in the experiments, our agent often does not survive past the first maze. This may be due to an imbalance in the game, or because ghosts do not always capture Pac-Man when using our simulation strategy. To improve the simulation strategy further, the knowledge rules of fast rule-based agents from competitions could be used if they performed well. Moreover, the rules could be optimized to increase the number of simulations each turn.

*Ms Pac-Man* is an interesting subject for AI research based on many of its characteristics. Agents need to consider both long-term and short-term goals. Moreover, its real-time nature makes it hard for algorithms such as MCTS that have to consider a multitude of simulations. Based on observations, our agent made between 200 and 400 simulations for each search. Investigating domain-independent techniques and enhancements for working with such games could lead to a better understanding of other real-time domains.

## REFERENCES

- [1] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Machine Learning: ECML 2006*, ser. Lecture Notes in Artificial Intelligence, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., Cambridge, MA, USA: MIT Press, 2006, vol. 4212, pp. 282–293.
- [2] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Computers and Games (CG 2006)*, ser. Lecture Notes in Computer Science, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., Berlin, Germany: Springer-Verlag, 2007, vol. 4630, pp. 72–83.
- [3] C. Browne et al., "A survey of Monte-Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
- [4] R. K. Balla and A. Fern, "UCT for tactical assault planning in real-time strategy games," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, C. Boutilier, Ed., 2009, pp. 40–45.
- [5] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Monte Carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 234–241.
- [6] G. Zhu, "Real-time elective admissions planning for health care providers," M.S. thesis, Schl. Comput. Sci., Univ. Waterloo, Waterloo, ON, Canada, 2013.
- [7] "Ms Pac-Man Competition, Screen-Capture," [Online]. Available: <http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html> May 2012
- [8] 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.
- [9] A. M. Alhejaly and S. M. Lucas, "Evolving diverse Ms. Pac-Man playing agents using genetic programming," in *Proc. IEEE Workshop Comput. Intell.*, 2010, DOI: 10.1109/UKCI.2010.5625586.
- [10] S. M. Lucas, "Evolving a neural network location evaluator to play Ms. Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, 2005, pp. 203–210.
- [11] D. Robles and S. M. Lucas, "A simple tree search method for playing Ms. Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, P. Luca Lanzi, Ed., 2009, pp. 249–255.
- [12] S. Samothrakakis, D. Robles, and S. M. 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.
- [13] B. K. B. Tong and C. W. Sung, "A Monte-Carlo approach for ghost avoidance in the Ms. Pac-Man game," in *Proc. Int. IEEE Consumer Electron. Soc. Games Innovat. Conf.*, 2010, DOI: 10.1109/ICEGIC.2010.5716879.
- [14] B. K. B. Tong, C. M. Ma, and C. W. Sung, "A Monte-Carlo approach for the endgame of Ms. Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, S.-B. Cho, S. M. Lucas, and P. Hingston, Eds., 2011, pp. 9–15.
- [15] N. Ikehata and T. Ito, "Monte-Carlo tree search in Ms. Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, S.-B. Cho, S. M. Lucas, and P. Hingston, Eds., 2011, pp. 39–46.
- [16] R. Thawonmas and T. Ashida, "Evolution strategy for optimizing parameters in Ms Pac-Man controller ICE Pambush 3," in *Proc. IEEE Conf. Comput. Intell. Games*, G. N. Yannakakis and J. Togelius, Eds., 2010, pp. 235–240.
- [17] "ICE gUCT (nqkien) Simulation Based Ghost Team Controller Report," [Online]. Available: <http://cec11.pacman-vs-ghosts.net/viewReport.php?id=10> Jul. 2012
- [18] "ICEgUCT\_CIG11 Simulation Based Ghost Team Controller Report," [Online]. Available: <http://cig11.pacman-vs-ghosts.net/report.php?id=44> Jul. 2012
- [19] "ICEpAmbush\_CIG11 Simulation Based Ms Pac-Man Controller Report," [Online]. Available: <http://cig11.pacman-vs-ghosts.net/report.php?id=50> Jul. 2012
- [20] K. Q. Nguyen and R. Thawonmas, "Applying Monte-Carlo tree search to collaboratively controlling of a Ghost Team in Ms Pac-Man," in *Proc. IEEE Int. Games Innovat. Conf.*, 2011, pp. 8–11.
- [21] K. Q. Nguyen and R. Thawonmas, "Monte-Carlo tree search for collaboration control of ghosts in Ms. Pac-Man," *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 1, pp. 57–68, Mar. 2013.
- [22] "CEC11 Ms Pac-Man vs Ghosts Competition Rankings," [Online]. Available: <http://cec11.pacman-vs-ghosts.net/rankings.php> Jul. 2012
- [23] T. Pepels and M. H. M. Winands, "Enhancements for Monte-Carlo tree search in Ms Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 265–272.
- [24] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Math. Natural Comput.*, vol. 4, no. 3, pp. 343–357, 2008.
- [25] 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.
- [26] 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.
- [27] 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.

- [28] L. Kocsis and C. Szepesvári, “Discounted UCB,” in *Lectures of PASCAL Second Challenges Workshop*, 2006 [Online]. Available: <https://www.lri.fr/~sebag/Slides/Venice/Kocsis.pdf>
- [29] A. Garivier and E. Moulines, “On upper-confidence bound policies for switching bandit problems,” in *Algorithmic Learning Theory*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2011, vol. 6925, pp. 174–188.
- [30] J. Hashimoto, A. Kishimoto, K. Yoshizoe, and K. Ikeda, “Accelerated UCT and its application to two-player games,” in *Advances in Computer Games*, ser. Lecture Notes in Computer Science, H. J. van den Herik and A. Plaat, Eds. Berlin, Germany: Springer-Verlag, vol. 7168, pp. 1–12.
- [31] Z. Feldman and C. Domshlak, “Simple regret optimization in online planning for Markov decision processes,” *CoRR* vol. abs/1206.3382, 2012 [Online]. Available: <http://arxiv.org/abs/1206.3382>
- [32] D. Bertsekas, *Dynamic Programming and Optimal Control, Vol II: Approximate Dynamic Programming*, 4th ed. Belmont, MA, USA: Athena Scientific, 2012.
- [33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
- [34] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, “Decaying simulation strategies,” in *Proc. Workshop Gen. Intell. Game-Playing Agents*, Y. Björnsson and M. Thielscher, Eds., Beijing, China, 2013, pp. 22–30.
- [35] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Mach. Learn.*, vol. 47, no. 2–3, pp. 235–256, 2002.
- [36] N. R. Sturtevant, “An analysis of UCT in multi-player games,” in *Computers and Games*, ser. Lecture Notes in Computer Science, H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, Eds. Berlin, Germany: Springer-Verlag, 2008, vol. 5131, pp. 37–49.
- [37] “ICEP-FEAT-SPOOKS (Controller Report),” [Online]. Available: <http://s3.amazonaws.com/wcci12/reports/8/ICEP-feat-Spooks-report-8.zip> Aug. 2013
- [38] M. Brandstetter and S. Ahmadi, “Reactive control of Ms. Pac Man using information retrieval based on genetic programming,” in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 250–256.
- [39] A. M. Alhejaly and S. M. Lucas, “Using genetic programming to evolve heuristics for a Monte Carlo tree search Ms Pac-Man agent,” in *Proc. IEEE Conf. Comput. Intell. Games*, 2013, pp. 65–72.



**Tom Pepels** received the B.Sc. degree in knowledge engineering from Maastricht University, Maastricht, The Netherlands, in 2012, where he is currently working toward the M.Sc. degree in artificial intelligence.

In 2012, his Ms Pac-Man agent won the 2012 IEEE Conference on Computational Intelligence and Games (CIG'12, Granada, Spain) Ms Pac-Man Versus Ghosts Competition. He is also an independent software engineer and IT consultant in his hometown Maastricht, The Netherlands.



**Mark H. M. Winands** (M'14) received the Ph.D. degree in artificial intelligence from the Department of Computer Science, Maastricht University, Maastricht, The Netherlands, in 2004.

Currently, he is an Assistant Professor at the Department of Knowledge Engineering, Maastricht University. His research interests include heuristic search, machine learning, and games.

Dr. Winands serves as a Section Editor of the *International Computer Games Association (ICGA) Journal* and as an Associate Editor of the IEEE

TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES.



**Marc Lanctot** received the Ph.D. degree in artificial intelligence from the Department of Computer Science, University of Alberta, Edmonton, AB, Canada, in 2013.

Currently, he is a Postdoctoral Research Fellow at the Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands. His research interests include search and sampling in games.