A Simple Tree Search Method for Playing Ms. Pac-Man

David Robles and Simon M. Lucas, *Senior Member, IEEE*

***Abstract*— Ms. Pac-Man is a challenging game for software agents that has been the focus of a significant amount of research. This paper describes the current state of a tree-search software agent that will be entered into the IEEE CIG 2009 screen-capture based Ms. Pac-Man software agent competition. While game-tree search is a staple technique for many games, this paper is, perhaps surprisingly, the first attempt we know of to apply it to Ms. Pac-Man. The approach we take is to expand a route-tree based on possible moves that the Ms. Pac-Man agent can take to depth 40, and evaluate which path is best using hand-coded heuristics. On a simulator of the game our agent has achieved a high score of 40,000, but only around 15,000 on the original game using a screen-capture interface. Our next steps are focussed on using an improved screen-capture system, and on using evolutionary algorithms to tune the parameters of the agent.**

I. Introduction

Games have long been used as a test bed for Artificial Intelligence (AI) and Machine Learning (ML) algorithms. Traditionally, most successful game playing strategies have been achieved through clever programming an effective strategies, but machine learning techniques such as evolutionary algorithms are now often competitive, and in some cases, superior to hand-coded algorithms [1].

The most popular maze video game of all time is Pac-Man, and while it is no longer the newest or most advanced example of game development, it still provides a platform that is both simple enough for AI research and complex enough to require intelligent strategies for successful gameplay. In recent years, Ms. Pac-Man (a variant of the original game) has received attention from the Computational Intelligence (CI) community, and one of the reasons for this is the competitions that have been held in conferences such as the IEEE Symposium on Computational Intelligence and Games (CIG). These provide a level playing field to establish which CI techniques work best.

Several techniques from all CI paradigms have been applied to Pac-Man agents, such as Genetic Algorithms, Reinforcement Learning, Neural Networks, Fuzzy Systems, etc. However, hand-coded rule-based systems have proven to produce the highest scoring agents. Therefore, in contrast to previous research carried out in Pac-Man, where the main goal is exclusively to use the game as a platform for research with intelligence being an emergent property, the aim of this work is producing apparently intelligent behaviour using whatever techniques are appropriate to produce a high-scoring agent to participate in the 2009 IEEE Symposium on Computational Intelligence and Games (CIG) Ms. Pac-Man competition[1].

This paper describes the application of a tree search strategy for path finding to play Ms. Pac-Man in a simulator and in the original game (via screen capture). The rest of this paper is organised as follows: in Section II we review the previous research in Pac-Man, Section III provides details of our Ms. Pac-Man simulator and the screen capture adapter, Section IV describes our agent and the use of the tree for path finding, Section V presents details and results of our experiments and a comparison to successful agents from previous Ms. Pac-Man competitions, and Section VI provides a summary and conclusions.

II. Previous Work

Previous research in Pac-Man has been carried out in different versions of the game (e.g. in the original game, Ms. Pac-Man and customised simulators), hence an exact comparison is not possible, but we can have a general idea of the performances. Most of these works can be divided in two areas: agents that use CI techniques partially or completely, and controllers with hand-coded approaches used in previous competitions. We will briefly discuss the most relevant works in both areas.

A. Computational Intelligence approaches

One of the earliest studies with Pac-Man was conducted by Koza [2] to investigate the effectiveness of genetic programming for task prioritisation. This work utilised a modified version of the game, using different score values for the items and also a different maze. According to Szita and Lorincz [3], the only score reported on Koza's implementation would have been equivalent to approximately 5,000 points in their Pac-Man version.

Bonet and Stauffer [4] proposed a reinforcement learning technique for the player, using a very simple Pac-Man implementation. They used a neural network and temporal difference learning (TDL) in a 10 x 10 centred window, but using simple mazes with only one ghost and no power pills. Using complex learning tasks, they showed basic ghost avoidance.

Gallagher and Ryan [5] used a Pac-Man agent based on a simple finite-state machine model with a set of rules to control the movement of Pac-Man. The rules contained weight parameters which were evolved using the Population-Based Incremental Learning (PBIL) algorithm. They ran a simplified version of Pac-Man with only one ghost and no power pills, which takes away scoring opportunities in the

The authors are with the School of Computer Science and Electronic Engineering, University of Essex, Colchester, United Kingdom. (phone: +44-1206-872048; fax: +44-1206-872788; email: darobl@essex.ac.uk, sml@essex.ac.uk).

[1] http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html

978-1-4244-4815-9/09/$25.00 ©2009 IEEE

game and takes away most of the complexity of the game. However this implementation was able to achieve machine learning at a minimum level.

Gallagher and Ledwich [6] described an approach to developing Pac-Man playing agents that learn game play based on minimal screen information. The agents are based on evolving neural network controllers using a simple evolutionary algorithm. Their results showed that neuroevolution is able to produce agents that display novice playing ability with no knowledge of the rules of the game and a minimally informative fitness function.

Lucas [7] proposed evolving neural networks to evaluate the moves. He used Ms. Pac-Man because the ghosts behave with pseudo-random movement, and that eliminates the possibility of playing the game using path-following patterns. This work utilised a handcrafted input feature vector consisting of the distances from Pac-Man to each non-edible ghost, to each edible ghost, to the nearest pill, to the nearest power pill and to the nearest junction. His implementation was able to score around 4,780 points in over 100 games.

Szita and Lorincz [3] proposed a different approach to playing Ms. Pac-Man. The aim was to develop a simple rule-based policy, where rules are organised into action modules and a decision about which direction to move is made based on priorities assigned to the modules in the agent. Policies are built using the cross-entropy optimisation algorithm. The best performing agent obtained an average score of 8,186, comparable to the performance of a set of five human subjects played on the same version of the game, who averaged 8,064 points.

More recently, Wirth and Gallagher [8] used an agent based on an influence map model to play Ms. Pac-Man. The model captures the essentials of the game and showed that the parameters of the model interact in a fairly simple way and that it is reasonably straight-forward to optimise the parameters to maximise game playing performance. The performance of the optimised agents averaged a score of just under $7,000$.

However, it should be emphasised that many of the above quoted scores have been obtained on the different Pac-Man simulators, and therefore only provide the very roughest idea of relative performance.

B. Hand-coded approaches

RAMP [9] was the winner agent of the Ms. Pac-Man competition in the 2008 World Congress on Computational Intelligence, with a high score of $15,970$ points and an average of $11,166$. It is a rule-based agent that uses a set of 9 conditions, such as number of ghosts close, number power pills remaining, if there are edible ghosts in the maze, etc. It also uses a set of only 4 actions: eat nearby pills, eat edible ghosts, evade ghosts and run to power pill. Outside of the competition the authors reported a high score of $18,050$ after reaching Level 5.

More recently, ICE Pambush 2 [10] won the competition held in the IEEE Congress on Evolutionary Computation (CEC) 2009, with a maximum score of $24,640$ and an average of $13,059$. This controller does an excellent job of extracting the game objects from the screen capture, and as the name suggests, lures the ghosts to the power pills and then ambushes them. It moves Ms. Pac-Man based on path costs and it uses a list of typical conditions and actions used in Pac-Man agents, such as nearest ghost distance, nearest power pill, evade ghosts, etc.

III. Experimental Setup

For the aforementioned purposes in Section I and also to enable much faster evaluation, we implemented our own Ms. Pac-Man Simulator. Also, we built useful diagnostic tools that give us full control over the agent we developed, since many AI data structures (e.g. the tree of paths) have a natural visual representation that is far easier to comprehend in its natural format than to attempt to interpret the data from stack dumps and variable watch windows in the debugger [11].

In order to produce a highly competitive agent, our work can roughly be divided into four main tasks:

1) Create a *Ms. Pac-Man simulator* that mimics the original game.
2) Create the *agent* that controls Ms. Pac-Man.
3) Develop a *screen capture adapter* to extract the information from the original game into the game model in the simulator.
4) *Tune* the algorithms and parameters, so that any given agent can operate in similar manner in both the simulator and the original game (i.e. without modification in the code).

A. Ms. Pac-Man Simulator

Our simulator is written in Java in object-oriented style and is based on, but significantly different from Lucas' simulator [7]. The architecture uses the popular Model-View-Controller (MVC) design pattern, and it replicates the majority of the features of the original game (at least on a functional level). However, there are some differences between our simulator and the original Ms. Pac-Man, some of which may have a significant impact on the difficulty of the game. These are the main features of our simulator which differ from the original:

- The speed of both Ms. Pac-Man and the ghosts are identical (they differ in the original game depending on the ghost and on the level).
- Ms. Pac-Man does not slow down to eat pills. This has several consequences that diminish the performance of the agent. For example, when the same agent previously proven in the simulator plays in the original game (i.e. screen-capture mode), Ms. Pac-Man is constantly caught up by ghosts in situations considered safe in the simulator.
- The ghosts do not slow down in the tunnels.
- Bonus fruits are not present, as their inclusion plays a relatively minor contribution to the score of the game (i.e. our main goal).
- Eaten ghosts do not wait at the centre of the level, they are reinserted and resume play immediately.

- Unlike some other Ms. Pac-Man implementations, this simulator provides an additional life at $10,000$ points. This provide us with a better idea of the comparability of the scores between the simulator and the original game.
- The edible time of the ghosts is replicated as in the original game, different in every level.
- Perhaps the most significant of all: the ghost behaviours are not the same as in the original game.

The levels are modelled as graphs of connected nodes. The first two levels of the game were replicated, and each level contains specific information regarding the pills, power pills, edible time of the ghosts, and starting positions for Pac-Man and the ghosts. On any given level, each node has two, three or four neighbouring nodes depending on whether it is in a corridor, L-turn, T-junction or a crossroads. Also, each node contains a *pill type* (empty, pill, power pill), representing whether a node is empty, if contains a pill, or a power pill. After the levels are created, the same algorithm used by Lucas [7] is run to compute the shortest-path distance between every node in the level and every other node in the same level. These distances are stored in a look-up table on each level to allow fast computation of important game features.

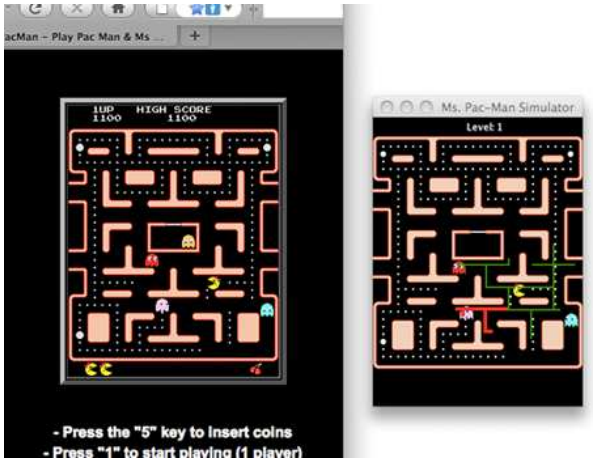

Fig. 1. The agent (right) controlling Ms. Pac-Man in the original game (left) via direct screen capture.

One of the problems faced while making the simulator was programming the motion of the ghosts. It is hard to replicate the actual behaviour of each ghost in Ms. Pac-Man. Although their behaviours have recently been described on some web sites, the descriptions are unclear in places, and we have not yet had time to implement them. Nonetheless, we observed that some ghosts are more aggressive than others, and their aggressiveness increases when Ms. Pac-Man is completing a level. To satisfy this characteristic, we chose a probabilistic approach to approximate their behaviours, by assigning to each ghost a probability of going towards Ms. Pac-Man on the shortest path. This probability is weighted according to the number of pills remaining in the current level to produce a similar behaviour as in the original game (i.e. ghosts become more aggressive at the end of the level). These probabilities are shown in Table I, where α is the fraction of pills that have been eaten on that level (i.e. so it starts at zero for each level).

TABLE I
GHOSTS' PROBABILITIES OF CHASING MS. PAC-MAN

Ghost	Probability
Blinky (Red)	0.9
Inky (blue)	$\alpha(0.85)$
Pinky (Pink)	$\alpha(0.80)$
Sue (Orange)	$\alpha(0.75)$

In addition, most of the rules in the game are satisfied in the simulator. The ghosts keep the same direction (i.e. can not reverse), and they reverse approximately every 20 seconds (though in the original game the ghosts are based on state machines and reversal is triggered by a state change rather than being so directly time related). Each game cycle is approximately 50 milliseconds. Also, just after Pac-Man has eaten a power pill all the ghosts reverse.

B. Screen Capture Adapter

The essence of the Ms. Pac-Man competition is that the input to the software agent is simply the real-time video output from the original game[2] (i.e. without any access to details of the software). There are a few screen capture controllers already available to build on, such as Lucas' screen capture kit[3], or the used by ICE Pambush 2 (the winning agent of the IEEE CEC 2009 Ms. Pac-Man Competition) [10], which does a very good job of extracting the game objects from the screen capture.

These screen capture controllers place the extracted information in their own representations of the Ms. Pac-Man game. They include the maze type, elements such as pills, power pills, state of the ghosts, position of Ms. Pac-Man, walls, empty corridors, etc. As the intention is to use the same agents in both the simulator and the original game, we decided to create a screen capture adapter to the game model in the simulator, and it was built upon the aforementioned screen capture controllers. This enables any agent to play in the simulator and in the original game without any changes in the code. We identified two fundamental tasks that must be done for the screen capture adapter to work properly:

1) *Effective detection*. The extracted objects (i.e. Ms. Pac-Man, ghosts, pills, power pills, etc.) must be detected in their exact locations with their current state (e.g. a ghost can be edible or non-edible).
2) *Efficient detection*. The extraction of the objects has to be fast enough to allow the agent to respond with a direction to control Ms. Pac-Man, e.g., a small delay in the detection can keep Pac-Man from cutting corners or be eaten by ghosts for slow reactions in ghost reversals.

[2]http://www.webpacman.com/
[3]http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html

The image processing of the captured image is carried out at every cycle of the Ms. Pac-Man agent decision process. This happens approximately every 80 milliseconds. First however, it begins by trying to detect the game on the screen automatically. It takes a full screenshot (i.e. not just the game), and reads every pixel of the image until it detects the colour of the top border of the level (so far we implemented only Level 1 and 2). It saves the position of the top left corner and then it repeats the following operations:

1) Captures the game to an image.
2) Detects the game objects (e.g. Ms. Pac-Man and ghosts).
3) Places the objects in the game model of the simulator.
4) Detects the pills and power pills.
5) Places the pills and power pills in the specified nodes of the model.
6) The agent executes the tree search algorithm, it finds the best path, and calculates the direction to move to that path.
7) Send the keystrokes to move to the desired direction using the Java Robot.

Figure 1 shows the agent under test controlling Ms. Pac-Man using the direct screen capture mode of operation. The Ms. Pac-Man game is shown to the left, and the extracted game objects are shown to the right.

The game is captured approximately 12 times per second, but the exact number depends on the speed of the computer and the amount of processing performed by the generation and evaluation of the tree. An interesting issue is that when testing the agent using the adapter in early stages of development, Ms. Pac-Man was missing the turns quite frequently. The first reason to come to our minds was either slow screen capture or slow tree processing. However, the problem was the inappropriate handling for sending key events, as the controller was sending press/release events on every cycle of the game, instead of sending a single key press event, and then a key release and a key press event only when a new direction was desired. This simple approach largely fixed the problem of missing the turns.

We consider that our screen capture adapter does a very good job extracting the game objects, however, there are key issues at the moment of the writing that we plan to correct before the competition:

- When a ghost is crossing a tunnel the screen capturer does not detect it, and Ms. Pac-Man behaves as if the tunnel was ghost-free.
- When the ghosts are in the centre of the maze (e.g. after being eaten), they are not detected at all. This causes a few deaths since the Ms. Pac-Man controller has no idea when they will reappear in the maze, and if Ms. Pac-Man passes next to the door when they are reappearing she will be eaten.

The authors tested both the simulator and the original game to asses the difficulty of the game. For these tests the games were played at normal speed for 20 games per person. Despite key differences between both games (e.g. Ms. Pac-Man does not slow down to eat pills) the scores were quite similar. In the simulator the authors scored an average of $15,176$ points, with a lowest score of $4,750$ points and best score of $34,100$ over 20 games. When playing in the original game, the average score obtained was $21,063$ points, with a lowest score of $10,590$ and a top score of $34,760$. The simulator seemed harder than the original, due to the fact that ghosts behave more randomly than in the original, and that is displayed in the inconsistent scores in the simulator. However, when the agents were tested both in the simulator and the original game, the results showed the opposite, since the agent's scores in the simulator were 30% higher. There are two possible reasons for this. It could be that despite it being easier for us, the original game is harder for our agent. At present though, it seems more likely to be due to some remaining problems with the screen capture interface. For example, the agent sometimes misses a turning or gets caught turning back into a single ghost when it has a clear escape route.

IV. Agent Design

The Ms. Pac-Man agent operates with the use of a tree, evaluating all the possible paths that can be taken from the current position. On every game cycle a tree is generated, taking the current node of Ms. Pac-Man as the root, and adding recursively the children of every node to the tree, until a maximum height is reached. Every node in the tree contains information about the type of content in the node, such as empty, pill, power pill, Ms. Pac-Man, ghost or edible ghost. It is important to note that a tree node in our implementation can have only one type of content (e.g. it can not have a pill and a ghost). Although it would be relatively simple to add multiple information to a tree node in the simulator, this approach of only one type was taken because with the screen capture adapter is hard (though not impossible) to identify a node with multiple content, since in a single screen-shot of the game a ghost hides a pill or power pill. Although it would clearly be possible to aggregate information over a sequence of frames to re-create the exact true game-state, our screen-capture engine is not yet this sophisticated.

After the tree has been fully created, a simple tree search algorithm is run to identify the paths that can be taken from the current position, independently if they are high-scoring or safe paths. With a tree of height of 40 nodes, the algorithm usually finds between 4 to 12 paths, depending on the position in the maze. The algorithm also counts for convenience the number of empty nodes, pills, power pills, ghosts, edible ghosts for each path of the tree. With this information set, the next step is to determine if a path is *safe* or *non-safe*. A safe path is defined as a path that can be reached to the last node with certainty, taking into account only the information from the tree. To determine the safeness of a path, we use the following procedure:

1) If a path does not contain a ghost and there are no ghosts in the tree, it is considered a safe path.
2) If a path contains a ghost it is considered non-safe. We did not take into account the direction of the ghost,

although this is an interesting feature to consider in future research.

3) If a path does not contain a ghost, but there is at least one ghost in the tree, the path is evaluated to see if Pac-Man can reach the first safe node in that path, i.e., a node of the path where the distance from the node to Pac-Man is less than the distance from the closer ghost to that node.

Figure 2 shows an example of a tree generated with a maximum height of 10 nodes. Empty nodes are shown non-filled, nodes with pills are shown as filled nodes, and the ghosts and Ms. Pac-Man are represented with their respective images. The tree has two reachable paths: C and D. Paths A, B, E and F are considered non-safe because they contain a ghost. Also, path E can be reached if the ghost moves in a direction opposite to Ms. Pac-Man, but since we do not consider the directions of the ghosts, it is considered non-safe.

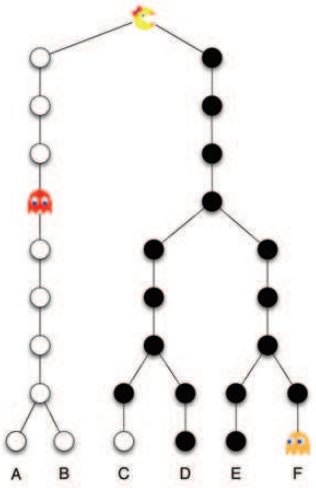

Fig. 2. Empty nodes are shown non-filled, nodes with pills are shown as filled nodes, and the ghosts and Ms. Pac-Man are represented with their respective images.

A. Path selection

Path selection is performed on every game cycle in the simulator, and in every screen capture in the original game. With each path set as either safe or non-safe, we can simply choose a path randomly and obtain reasonable performance. For example, in Figure 3 the red (thick) paths are non-safe (they lead to a ghost) and the green (thin) paths are safe. However, in order to optimise the path selection, we need to determine which is the best path in two specific situations: 1) when there is more than one safe path; and 2) when there are no safe paths at all. In the first situation, it is desirable to take the path with the highest value, and in the second it is desirable to select the best non-safe path (i.e. the least bad), which can be the path where the distance to the ghost is the furthest. The question that emerges is, how do we define the a value for each path? A simple approach is to select the path with a power pill, or the one with most pills.

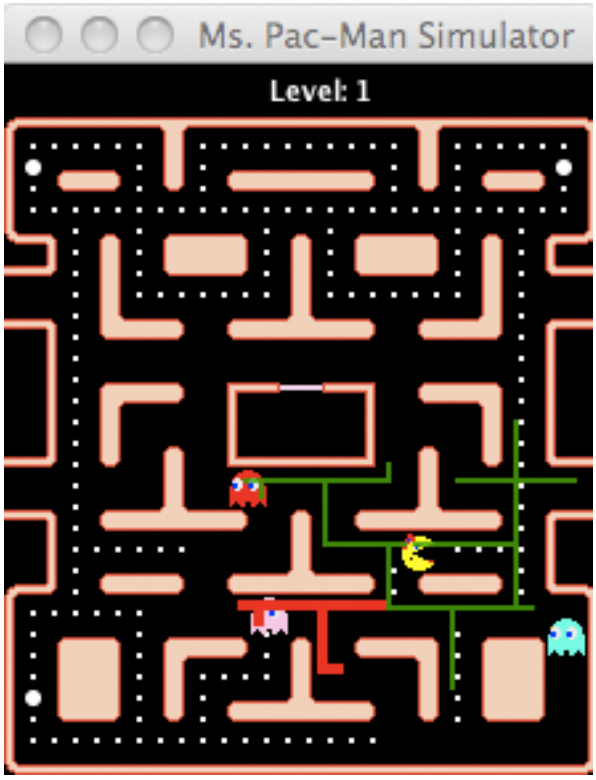

Fig. 3. Tree generated to evaluate the possible paths to take.

So we decided to have another way of selecting paths more efficiently, and we hand-coded an algorithm to select a path depending on the number of pills, power pills, position of path on the tree.

Figure 4 shows the same situation as in Figure 2, but here we can see safe and non-safe paths easily.

V. Results

We defined 3 strategies that the agent can use for path selection:

- *Rand-Safe-Path:* Selects a random safe path.
- *Most-Pills-Safe-Path:* If there is a path with a power pill, selects it. If not, it picks the path with most pills.
- *Hand-Coded-Select:* Selects the best path according to a hand-coded value function that takes the number of pills, number of power pills, and location of the end of path. Locations with a vertical position toward the middle of the maze are preferred on the basis of our observation that they tend to be safer. The utility of this positional feature has not been confirmed beyond doubt, but on our tests it did improve the average score. The agent also uses information outside of the tree. This strategy waits close to the power pills for ghosts to approach, and then it eats the power pills and attempts to eat them. Also, it chases ghosts located not only in the tree, but in any other place of the level. This behaviour was also a significant feature of ICE Pambush 2 [10].

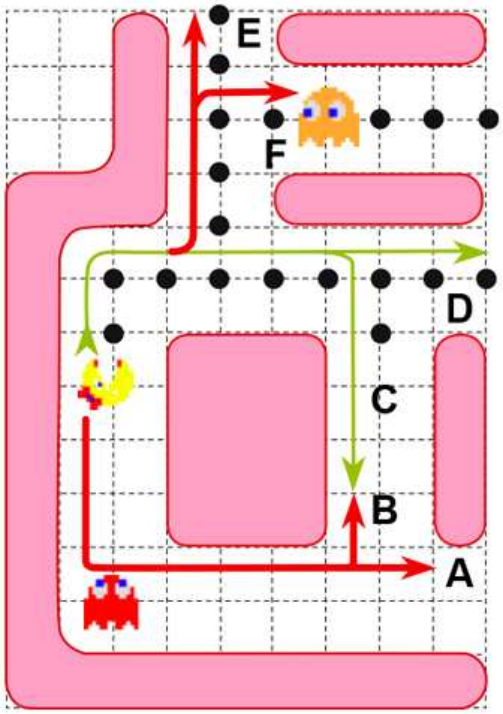

Fig. 4. Safe and Non-Safe Paths.

A. Experiments with the Ms. Pac-Man Simulator

The experiments were first performed in the simulator for 100 games for each strategy. The results showed very disparate scores, and the reason for this is probably the behaviour of the ghosts, i.e., random behaviour at the beginning of the levels. In the *Rand-Safe-Path* strategy, the average score obtained was 3,068, with a lowest score of 710 and a highest score of 6,820. This strategy was able to obtain good scores for a random controller, and the reason is that the paths that Pac-Man can take are already considered safe or non-safe, which allows the agent to survive for longer compared to the other strategies since Pac-Man takes random safe-paths. However, it often fails to complete a level since it has no incentive to eat pills.

TABLE II

RESULTS FROM THE MS. PAC-MAN SIMULATOR. THE STANDARD ERROR (S.E.) GIVES A GOOD IDEA OF THE CONFIDENCE INTERVALS.

Controller	Min	Max	Mean	s.e.
Rand-Safe-Path	710	6820	3068	146
Most-Pills-Path	1720	22370	8894	470
Hand-Coded-Sel	4270	43170	14757	782

The results for the *Most-Pills Path* strategy were much better than *Rand-Safe-Path* controller, with an average of 8,894. This strategy was able to clear the levels more often, since it is always going towards the path with more pills. It is interesting to note that the difference between the lowest score (1,720) and the higher score (22,370) is large. The *Hand-Coded-Sel* strategy offers the best performance with a high score of over $40,000$. The average score improved almost 6,000 points from the *Most-Pills-Path* controller, due to the use of simple but effective aforementioned rules.

B. Experiments in the original game

The same experiments with the three strategies were performed in the original game (using screen capture adapter), but for only 20 games, since it is not possible to run just the model of the game. As expected, the scores were much lower and there are several reasons why the agents perform worse in this game. One is that in the simulator Ms. Pac-Man does not slow down to eat pills and in the real game it does, and this produces many deaths that would not happen in the simulator. In addition, despite the reasonably accurate extraction of the game objects, the issues mentioned in Section III still cause a decrease in the scores.

TABLE III

RESULTS FROM THE ORIGINAL GAME (VIA SCREEN CAPTURE). THE STANDARD ERROR (S.E.) GIVES A GOOD IDEA OF THE CONFIDENCE INTERVALS.

Controller	Min	Max	Mean	s.e.
Rand-Safe-Path	380	1900	820	34
Most-Pills-Path	2110	8050	3878	168
Hand-Coded-Sel	3570	15280	9630	346

With the *Rand-Safe-Path* strategy the scores were very poor, with an average of 820 only, compared to 3068 obtained in the simulator. One of the reasons is that in the simulator the agent takes decisions once the ghosts are very close to Ms. Pac-Man (e.g. at a distance of one node), and with the screen capture adapter these small distances are not detected with sufficient precision. Also, the behaviour of the ghosts is much harder for the agent, since it loses in the traps very often (despite the fact that the simulator was harder in the play-tests undertaken by the authors). In the *Most-Pills Path* strategy the average score was 3,878 which is just the half of the achieved in the simulator. This strategy also had the issues of the previous strategy, but at least it was able to finish the levels at times (although with low scores). Finally, the best strategy from the experiments with the simulator was performed in the screen capture version with good results. As expected, the results were much lower than in the simulator with an average score of 9,630.

VI. CONCLUSIONS

The aim of this paper was to develop a high-scoring agent to participate in the 2009 IEEE Symposium on Computational Intelligence and Games (CIG) Ms. Pac-Man competition. The approach taken was to use a simple tree search algorithm to explore the set of possible paths that Ms. Pac-Man could take from her current position. This approach is not a proper game-tree search, since it does not consider movement of the ghosts, or changes in ghost state that would be caused by eating a power pill for example. Nonetheless, despite these obvious limitations, it performed surprisingly

well on our simulator of the game. We are currently considering a full game tree search, or Monte Carlo based tree search that will overcome these limitations. In particular, Monte Carlo algorithms seem very promising and have achieved great success in Go [12] [13], and in General Game Playing [14]. The latest version of our simulator incorporates an extremely compact representation of the game-state making it well suited to Monte-Carlo methods.

Finally, our aim is to ensure that good performance on the simulated game transfers to good performance when playing the original game in screen-capture mode. In order to achieve these we are currently increasing the fidelity of the simulator, and the performance of the screen-capture kit.

REFERENCES

[1] S. M. Lucas and G. Kendall, "Evolutionary computation and games," *IEEE Computational Intelligence Magazine*, February 2006.

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

[3] I. Szita and A. Lorincz, "Learning to play using low-complexity rule-based policies," *Journal of Artificial Intelligence Research*, vol. 30, no. 1, pp. 659–684, December 2007.

[4] J. S. D. Bonet and C. P. Stauffer, "Learning to play pac-man using incremental reinforcement learning," *Proceedings of the Congress on Evolutionary Computation*, 1999.

[5] M. Gallagher and A. Ryan, "Learning to play pac-man: An evolutionary, rule-based approach," in *IEEE Symposium on Computational Intelligence and Games*, 2003, pp. 2462–2469.

[6] M. L. Marcus Gallagher, "Evolving pac-man players: Can we learn from raw input?" in *IEEE Symposium on Computational Intelligence and Games*, 2007.

[7] S. M. Lucas, "Evolving a neural network location evaluator to play ms. pac-man," *IEEE Symposium on Computational Intelligence and Games*, pp. 203–210, 2005.

[8] N. Wirth and M. Gallagher, "An influence map model for playing ms. pac-man," *IEEE Symposium on Computational Intelligence and Games*, 2008.

[9] A. Fitzgerald, P. Kemeraitis, and C. B. Congdon, "Ramp: A rule-based agent for ms. pac-man," in *World Congress on Computational Intelligence*, 2008.

[10] R. T. Hiroshi Matsumoto, Chota Tokuyama, "Ice pambush 2," in *http://cswww.essex.ac.uk/staff/sml/pacman/cec2009/ICEPambush2.pdf*, 2008.

[11] S. Rabin, *AI Game Programming Wisdom*. Charles River Media, Inc., 2002.

[12] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of uct with patterns in monte-carlo go," INRIA, Tech. Rep. Technical Report 6062, 2006.

[13] C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, "The computational intelligence of MoGo revealed in Taiwan's Computer Go tournaments," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, pp. 73 – 89, 2009.

[14] Y. Bjornsson and H. Finnsson, "CadiaPlayer: A simulation-based general game player," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, pp. 4 – 15, 2009.

[15] J. L. Thomas Thompson, Lewis MacMillan and A. Andrew, "An evaluation of the benefits of look-ahead in pac-man," in *IEEE Symposium on Computational Intelligence and Games*, 2008.

[16] J. E. Laird and M. van Lent, "Human-level AI's killer application: Interactive computer games," in *American Association for Artificial Intelligence*, 2001.