MS PAC-MAN VERSUS GHOST TEAM CEC 2011 Competition

Philipp Rohlfshagen
School of Computer Science
and Electronic Engineering
University of Essex
Colchester CO4 3SQ, UK
Email: prohlf@essex.ac.uk

Simon M. Lucas
School of Computer Science
and Electronic Engineering
University of Essex
Colchester CO4 3SQ, UK
Email: sml@essex.ac.uk

***Abstract*—Games provide an ideal test bed for computational intelligence and significant progress has been made in recent years, most notably in games such as GO, where the level of play is now competitive with expert human play on smaller boards. Recently, a significantly more complex class of games has received increasing attention: real-time video games. These games pose many new challenges, including strict time constraints, simultaneous moves and open-endedness. Unlike in traditional board games, computational play is generally unable to compete with human players.**

One driving force in improving the overall performance of artificial intelligence players are game competitions where practitioners may evaluate and compare their methods against those submitted by others and possibly human players as well. In this paper we introduce a new competition based on the popular arcade video game MS PAC-MAN: MS PAC-MAN VERSUS GHOST TEAM.

The competition, to be held at the Congress on Evolutionary Computation 2011 for the first time, allows participants to develop controllers for either the Ms Pac-Man agent or for the Ghost Team and unlike previous MS PAC-MAN competitions that relied on screen capture, the players now interface directly with the game engine. In this paper we introduce the competition, including a review of previous work as well as a discussion of several aspects regarding the setting up of the game competition itself.

***Index Terms*—Computational Intelligence, Games, Game Competition, Ms Pac-Man, Predator-Prey**

I. INTRODUCTION

The field of computational intelligence (CI) has had noticeable success in recent years in developing computational tools that may compete with human expertise in a variety of domains. One such domain is games, including traditional boardgames such as CHESS or GO and real-time video games such as UNREAL TOURNAMENT. Games pose an interesting challenge, both academically and commercially and have been subject to a long-established research effort.

Academically, games provide an ideal test bed for the development and testing of new techniques and technologies: games are defined by an explicit set of rules and the goal of playing a game is usually defined unambiguously by the game's score or outcome. Games are also immensely flexible and vary greatly in complexity from single player puzzles to two-player boardgames to massively multi-player real-time video games. Furthermore, it is important to note that techniques developed specifically for game playing may often be transferred easily to other domains, greatly enhancing the scope with which such techniques may be used. Monte Carlo Tree Search [9], for example, allowed for a breakthrough in human-competitive play in the classic board game GO; recently, the same technique has been applied successfully to other domains such as scheduling [13].

Undoubtedly, there is significant commercial interest in developing strong game AI as well. The video game software industry in the USA alone is worth an annual turnover of US$ 4.9 billion (2009) with a growth rate of 10.6% for the period 2005-2009 (the growth rate for the US economy as a whole was 1.4% for the same period) [20]. Here, the goal of AI agents is usually not to achieve the strongest possible play but to optimise the overall playability of the game: human players need to be engaged at the right level of difficulty to make the game appealing, a task that has proven difficult as Yannakakis and Hallam [26, p 119] point out: "the increasing number of multi-player online games (among others) is an indication that humans seek more intelligent opponents and richer inter-activity."

A solution to this dilemma is the development of stronger non-player characters (NPCs) that do not rely primarily on classical game artificial intelligence (AI) methods such as scripting, triggers and animations.[1] Game competitions provide an ideal testbed for practitioners to further the development of NPCs that play a game *intelligently* and in this paper we introduce a new game competition based on the popular arcade game MS PAC-MAN: previous MS PAC-MAN-competitions required participants to develop AI controllers for the Ms Pac-Man character. The MS PAC-MAN VERSUS GHOST TEAM-competition allows participants for the first time to also develop multi-agent controllers for the ghost team. This paper outlines the scope, rules and technical specifications of this competition.

First we introduce the game MS PAC-MAN in section II

[1]As Ahlquist and Novak [1, p 4] point out, computer science AI and game AI are only distantly related: the former is about substance, whereas the latter is about appearances. The limitations of game AI have are discussed, for instance, in [23].

TABLE I
SUMMARY OF PREVIOUS MS PAC-MAN-SCREEN-CAPTURE COMPETITIONS.

Venue	Winners	Agent Name	Score
CEC'07	N/A	Default Agent	2,269
WCCI'08	A. Fitzgerald et al.	N/A	15,970
CEC'09	H. Matsumoto et al.	ICE Pambush 2	24,640
CIG'09	H. Matsumoto et al.	ICE Pambush 3	30,010
CIG'10	E. Martin et al.	Pac-mAnt	21,250

and review previous research related to developing Ms Pac-Man controllers and predator-prey models in section III. We present the rules of the game in section IV, including possible restrictions imposed upon the controllers. In section V we discuss the actual implementation of the game and then present several example controllers in section VI. Finally, section VII discusses some of the technical details regarding the setup of the competition before the paper is concluded in section VIII.

II. MS PAC-MAN VERSUS GHOST TEAM

One of the earliest commercially successful games, now a classic and still played all over the world, is PAC-MAN, an arcade game developed in 1980 by Toru Iwatani. The best known variant of the game is MS PAC-MAN, released in 1981, which introduced a female character, new maze designs and several gameplay changes. Screenshots of the game are shown in Figure 1: Ms Pac-Man needs to navigate around the maze, eating pills (sometimes called pellets) for points while trying to avoid the four ghosts who strive to eat Ms Pac-Man. The four power-pills in the corners of the maze allow Ms Pac-Man to eat the ghosts for a limited time (the ghosts turn blue) to gain additional points. In the arcade version of the game, the Ms Pac-Man character is controlled by the player using a 4-way joystick, allowing Ms Pac-Man to move north, south, east and west. If the joystick is kept in neutral or whenever an illegal move is chosen, the previous action is repeated. If Ms Pac-Man encounters a wall, she remains stationary until the next legal joystick event occurs.

The probably most significant change of MS PAC-MAN over the original is the design of the ghost team which now has elements of randomness that make the game more engaging: while the maximum possible score for PAC-MAN was achieved in 1999, new high-scores for MS PAC-MAN are still being set: a new record of 921,360 points was set by Abdner Ashman in 2006 according to www.twingalaxies.com. Scores obtained by AI controllers, on the other hand, are significantly inferior: MS PAC-MAN has featured in several previous competitions, most notably the MS PAC-MAN-Screen Capture competition, held annually since 2007 at various CI conferences. The scores obtained at these competitions are shown in Table I. There are at least two reasons for this significant discrepancy.

First, the state of the game in previous competitions is obtained by a screen reader that parses the game into a secondary representation. Controllers thus need to deal not only with the dynamics of the game but must also account for possible delays and missing information. Although such requirements are commonplace in many domains, especially those for which CI methods are frequently required in the first place, eliminating these uncertainties by interfacing directly with the game engine allows practitioners to fully concentrate on the game itself. An efficient simulator of the game has thus been developed for the MS PAC-MAN VERSUS GHOST TEAM-competition that may be accessed efficiently using simple software interfaces (see sections IV and V).

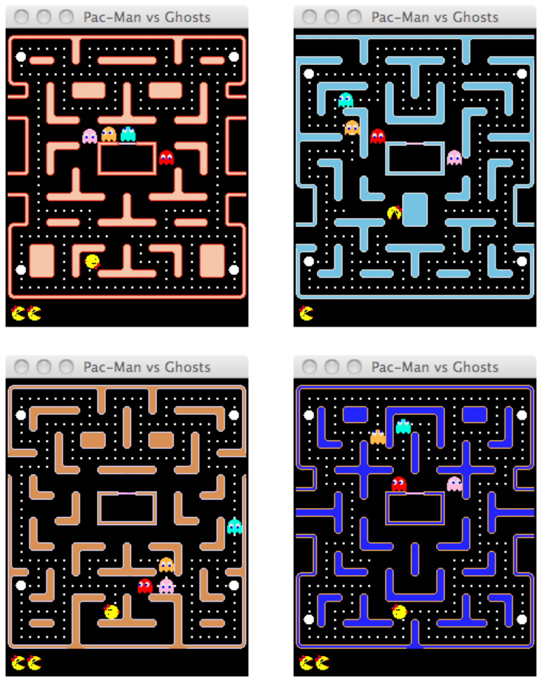

Fig. 1. Screen captures of the different levels (left-to-right, top-to-bottom, levels 1-4) of MS PAC-MAN: Ms Pac-Man (yellow) consumes pills for points (small white dots) while the ghosts (red: Blinky, pink: Pinky, green: Inky and brown: Sue) attempt to eat her; the large white dots in the corners are the power pills.

Second, the real-time element of the game poses a particular challenge and makes a crucial difference to the type of technologies that may be employed. Although time also plays an important role in turn-taking board games such as CHESS, the duration granted to determine a move in real-time video games are usually orders of magnitude smaller. This makes MS PAC-MAN a particularly challenging test bed for developing novel AI techniques which need to be both theoretically and practically sound.

The MS PAC-MAN VERSUS GHOST TEAM competition introduces an additional aspect: while MS PAC-MAN is traditionally viewed from the perspective of Ms Pac-Man (i.e., the ghosts are assumed to be NPCs), participants are now allowed to develop controllers for the ghost team as well. This provides an excellent environment for testing multi-agent strategies. Here the aim could be two-fold: practitioners may aim to optimise the playability of the game, or to minimise the score obtained by Ms Pac-Man. It is this latter objective that provides the focus of this competition, simply because it is unambiguously defined. In the near future, however, especially with the participation of human players, the former objective may play an important role as well; measurements for the entertainment value of games already exist and may be utilised towards this goal (see, e.g., [26], [27]).

III. Recent Research

The popularity of Pac-Man and Ms Pac-Man sparked noticeable academic interest and this section reviews some of these research efforts. It is important to note that researchers have considered a variety of different variants of the original games (usually custom implementations or variants of existing implementations), making it difficult to compare the scores obtained by the AI controllers. We thus do not report such scores.

A. Pac-Man *and* Ms Pac-Man *Controllers*

One of the earliest studies related to Pac-Man is by Koza [10] who investigated the effectiveness of genetic programming (GP) for task prioritisation where the controller would choose amongst a set of predefined rules (also see [18], cited in [22]). Alhejali and Lucas [2] also employed GP to evolve a wide variety of Ms Pac-Man agents. In particular, the authors evolved a diverse set of behaviours for different version of the game, demonstrating that GP was able to evolve behaviours well suited for each variation.

Burrow and Lucas [4] analysed the learning behaviours of temporal difference learning (TDL) and evolutionary algorithms (EAs) in conjunction with function approximators: two features served as input to the function approximators (interpolated table and multi-layer perceptron) to produce an estimate of the value of moving to the candidate node. The experiments showed that is this particular case, the EA outperformed TDL.

Gallagher and Ryan [6] used a Pac-Man agent based on a simple finite-state machine with a set of rules to control the agent's movement depending on its current position. The rules consisted of weighted parameters that were evolved using the population-based incremental learning algorithm. Szita and Lorincz [21] also developed a simple rule-based policy, where rules are organised into action modules and a decision about the agent's next direction is made based on priorities assigned to the modules in the agent; policies were built using the cross-entropy optimisation algorithm.

Robles and Lucas [17] applied a tree search algorithm to the game: the approach taken was to expand a route-tree based on possible moves that Ms Pac-Man can take, up to a tree depth of 40. The best path was subsequently evaluated using hand-coded heuristics. Bell et al. [3] combined tree searches with rule-based systems, making use of Dijkstra's algorithm, to determine the next direction of Ms Pac-Man's. The authors also accounted for the movement of the ghosts to determine safe paths for Ms Pac-Man. Similarly, Oh and Cho [14] use a rule-based approached, also based on Dijkstra's algorithm. An evolved neural network was subsequently used to select rules for different situations in the game.

Evolved neural networks were also used in prior work by Lucas [11] to evaluate Ms Pac-Man's possible moves; simple path-following patterns are inappropriate in Ms Pac-Man due to the stochastic behaviour of the ghosts. The controller utilised a handcrafted input feature vector consisting of the distances of Ms 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. Likewise, Gallagher and Ledwich [5] employed minimal screen-capture information to develop Pac-Man agents capable of learning how to play the game: the agents are governed by neural network controllers evolved using a simple evolutionary algorithm. The results showed that neuro-evolution is able to produce agents that display novice playing ability without prior knowledge of the rules of the game as well as a minimally informative fitness function.

Samothrakis et al. [19] used Monte Carlo Tree Search (MCTS), using a 5-player max^n game tree (i.e., each ghost is treated as an individual player). Unlike traditional tree searches, MCTS is an anytime algorithm with an asymmetric tree structure, lending itself nicely to the real-time constraints of the game.

Finally, Wirth and Gallagher [22] designed agents based on influence maps: the model used captured the essentials of the game (e.g., desirable regions in the maze) and subsequently demonstrated that the model's parameters interact in a fairly simple way, making it possible to optimise the parameter values that maximise the agent's game playing performance.

B. *Ghost Team Controllers and Predator-Prey Models*

The number of ghost team controllers developed for Pac-Man or Ms Pac-Man is rather small compared to the number of Ms Pac-Man controllers. Nevertheless, many result have been obtained for the more general scenario of *predator-prey*, often by co-evolutionary means, that may be helpful to the participants of the competition.

In [23], Wittkamp et al. explore, as a proof of concept, the use of CI techniques for real time learning to evolve strategies for the ghost team. Using a neural network to control the ghosts, focus is on team-work development that makes use of continuous short-term learning to regularly update the strategy of the ghosts. Ultimately, this strategy attempts to exploit the weaknesses of Ms Pac-Man. Yannakakis and Hallam [24], [25] propose a generic approach to generate interesting interactive Pac-Man opponents. Here the focus is on the entertainment value of the game and not the raw performance of the ghost team: the authors demonstrate that their neuro-evolution learning mechanism (the actions of the ghosts are determined by a fully connected feedforward neural controller) is able to increase the Ms Pac-Man game's interest and to sustain it at high levels.

In a more general setting, Haynes et al. [8], [7] strive to generate programs for the coordination of cooperative autonomous agents in pursuit of a common goal. The authors consider a simple predator-prey pursuit game, noting that the problem is easy to describe yet extremely difficult to solve. An extension of GP was used to evolve teams of agents with different strategies for their movements. Similarly, Luke and Spector [12] consider different breeding strategies and coordination mechanisms for multi-agent systems evolved using GP. In particular, the authors are interested in the performance of homogenous and heterogenous teams: in a heterogeneous team of agents, each agent is controlled by a different algorithm whereas homogenous agents are all controlled by the same mechanism. The problem considered is called the Serengeti world, a toroidal, continuous 2-dimensional landscape, inhabited by gazelles and lions.

Finally, examples of recent work regarding predator-prey scenarios include Rawal et al. [16] and Rajagopalan et al. [15]. In both cases, the authors consider the co-evolution of simultaneous cooperative and competitive behaviours in a complex predator-prey domain. The authors propose an extended neural-network architecture to allow for incremental co-evolutionary improvements in the agents' performance. This mechanism demonstrated hierarchical cooperation and competition in teams of prey and predators. The authors further note that in sustained co-evolution in this complex domain, high-level pursuit-evasion behaviours emerge.

This review highlights the promise of neural-network controllers (possibly evolved or co-evolved) with a mixture of offline and online learning and the addition of predetermined rules. Important aspects to consider include the internal team dynamics (coordination, cooperation) as well as the behaviours towards group outsiders. In terms of the MS PAC-MAN VERSUS GHOST TEAM-competition, robust behaviour towards a wide range of different opponents may prove valuable during the round-robin stages. However, it should be noted that most studies reviewed above did not consider real-time constraints and it remains to be seen how the complexity of an approach fares given the limited computation time available between moves.

IV. MS PAC-MAN VERSUS GHOST TEAM: RULES

The previous section highlighted the variety of techniques used to design novel controllers for MS PAC-MAN. However, it remains difficult to compare the performances of the individual efforts as testing is mostly carried out on different variants of the game. The MS PAC-MAN VERSUS GHOST TEAM-competition attempts to address this issue, allowing all participants to work on the same game engine. The software used does not rely on screen-capture but instead is a reasonably accurate implementation of the original game that allows participants to interface directly with the game engine. This also allows the integration of arbitrary ghost behaviours, which is of course a necessary condition for the competition. The following outlines the rules of the game as used in the competition.

A. *Rules of* MS PAC-MAN

The game consists of four mazes in total (A, B, C and D), which are worked through in that order. When maze D is cleared, the game goes back to maze A and continues the same sequence until the game is over. Each maze contains a different layout with pills and power pills placed at specific locations. The player starts in maze A with three lives, and a single extra life is awarded when reaching 10,000 points. The goal of the Ms Pac-Man is to obtain the highest possible score by eating all the pills and power pills in the maze (and thus advancing to the next stage). Each pill eaten scores 10 points, each power pill is worth 50 points. The difficulty of clearing each maze is due to the four ghosts: Blinky (red), Pinky (pink), Inky (green) and Sue (brown). At the start of each level, the ghosts start in their lair in the middle of the maze and spend some idle time before entering the maze, starting their pursue of Ms Pac-Man. The time spent in the lair before joining the chase decreases as the player progresses to higher levels. Each time Ms Pac-Man is eaten by a ghost, a life is lost and Ms Pac-Man and the ghosts return to their initial positions.

There are four power pills in each of the four mazes, which, when eaten, reverse the direction of the ghosts and turn them blue; they may now be eaten for extra points. The score for eating each ghost in succession immediately after a power pill has been consumed starts at 200 points and doubles each time, for a total of $200+400+800+1600 = 3000$ additional points. Any ghost that has been eaten re-appears in the lair and emerges soon after, once again chasing Ms Pac-Man. If a second power pill is consumed while some ghosts remain edible, the ghost score is reset to 200; if the level is cleared while the ghosts remain edible, play continuous immediately to the next level. The more advanced the level, the shorter the edible time becomes, making the levels progressively more difficult and at an advanced stage, the ghosts do not turn blue at all (however, they still change direction). When the edible period runs out, the ghosts start flashing blue and white. The player (or controller) needs to be careful at this stage to avoid losing lives. When all the pills and power pills have been cleared, the game moves on to the next maze.

The goal for Ms Pac-Man is to maximise her score while the ghost team should strive to minimise that score. It should be noted that contestants may provide both a Ms Pac-Man controller and a ghost team controller, but these should not collude to provide an advantage to an opponent via self-sacrifice; this will be checked for either by examining the source code submitted or by preventing such entries to compete directly with one another.

B. *Deviations from the Original*

An attempt was made to preserve the details of the original game as much as possible. However, a few changes were made to simplify the game. At design time these were intended to have minimal impact on the game play:

- The speed of Ms Pac-Man and the ghosts are identical.
- Bonus fruits are omitted.

However, on observing a large number of games, the point relating to the speed differences does actually have a significant impact on the gameplay, and a more faithful implementation of these aspects is planned for a future competition: although one of the goals of this competition is to establish a consistent platform for the development of new AI controllers, some future deviations are to be expected once a sufficient number of entries have been compared. Finally, since the competition allows for controllers for both Ms Pac-Man as well as the ghost team, the original ghost team controller is absent from the game.

C. Ghost-Team-Specific Rules

There are no restrictions regarding the actions of Ms Pac-Man and movement in any direction not blocked by a wall is allowed at all times. For the ghost team, on the other hand, three restrictions apply: the first follows from the original game specifications and prevents a ghost from turning back on itself. In other words, a ghost may only choose its direction at a junction. The second restriction also follows from the original game: occasionally there is a global reversal event when all the ghosts suddenly change direction. In the original game this happens when particular conditions are met (such as a specific number of pills having been consumed). In our implementation a global reversal event can happen on any game tick with probability of 0.005. This adds an element of randomness to each game. Finally, the third restriction is a competition-specific one to allow for competitive game play: each level is limited to 2000 game ticks, after which the game moves on to the next level; Ms Pac-Man is rewarded the points of all remaining pills. This modification is to prevent game spoiling tactics where the ghosts continuously circle the last available few pills, thus preventing the level from being cleared. With the time-limit in place, the ghost team is forced to take a more proactive approach.

V. Implementation

The game is written entirely in Java and no additional software is required by the participants; the source code provided contains all classes required to get started in writing some hand-coded controllers, but does not include any software for neural networks or evolutionary algorithms. The software may be downloaded at www.pacman-vs-ghosts.net. Java was chosen for its popularity, extensive documentation and ease of implementation, allowing for a wider range of participants. Nevertheless, we hope to extend this competition to other programming languages in the near future.

The game may be executed in visual and non-visual mode, synchronously or asynchronously. The non-visual synchronous (non-threaded) mode is particularly useful for testing and may exceed 100 games per second (depending on the controller's complexity and ability; more able controllers lead to longer-lasting games). The competition will run the game in asynchronous (threaded) mode with each controller running in a separate thread: at each game tick the controllers (for Ms Pac-Man and the ghost team) are provided with the current state of the game. The game runs in real time with 25 game ticks per second and each controller may take as long as it wants to respond. This allows the controller to strike an appropriate balance between the quality of the decision (move) and the time taken to make that decision. Of course, the downside of taking too long is that the game state may be out of date by the time a decision is made. Each cycle, the game engine waits for 40ms and then updates its state based on the most recent outputs (i.e., directions) from each controller.

At each time step, the controller needs to return a direction for each of the agents controlled (i.e., a single action for Ms Pac-Man controllers and four actions of ghost team controllers). These actions are simply integers in the range 0-4 to indicate up, right, down, left and neutral respectively. The move chosen clearly depends on the state of the game which is supplied to each controller at every time step. The game state may be queried to obtain all relevant information, including the positions of the agents, the availability of pills and power pills and so on. It should be noted that the code is and will remain under active development and although we encourage participants to examine or even contribute towards the code, they should rely solely on the Java interfaces provided, which will not change. In the following we outline those interfaces and explain further details regarding the game's implementation. For the sake of simplicity and clarity, we have simplified some of the code, most notably removing brackets as well as some Java keywords and punctuation characters; furthermore, only the most relevant methods are included.

A. GameState and Maze

The state of the game is represented by *GameState.java* and is determined by the current maze, which has a specific layout (immutable part of the game state) and the variable factors such as the positions of all agents, information regarding the pills and power pills, whether the ghosts are currently edible, etc. The game state thus provides all the information required by the controllers to make an informed decision. The state of the pills and power-pills are stored in BitSets, allowing a very compact representation of this aspect of the game. However, this requires a bit more work to get meaningful information such as finding the nearest pill and hence utility methods are provided to allow agents to obtain such information without having to implement additional methods. The game state contains the following important methods:

```
GameStateInterface copy()
void next(int pacDir, int[] ghostDirs)
MazeInterface getMaze()
BitSet getPills()
BitSet getPowers()
MsPacMan getMsPacman()
Ghosts[] getGhosts()
int getLevel()
int getScore()
```

The use of bit-sets enables the entire state of the game to be represented in a few tens of byte, making this an excellent platform for tree-search based controllers where it may be necessary to efficiently copy the game state. This efficient encoding also enables very low communication overheads for network-based play. The mazes of the game are modelled as graphs of connected nodes and contains the following important methods:

```
int dist(Node a, Node b)
ArrayList<Node> getPowers()
ArrayList<Node> getPills()
ArrayList<Node> getMap()
Node[][] getNode2DArray()
Node getNode(int x, int y)
Node getNode(int index)
```

Each node has two, three or four neighbouring nodes depending on whether it is in a corridor, L-turn, T-junction or a crossroads. After the mazes have been created, a simple efficient algorithm is used to compute the shortest-path distances between all nodes for each of the mazes; to enable efficient computation, each node has a unique node index going from zero to $(n-1)$, where there are n nodes in the graph. These distances are stored in a look-up-table, and allow fast computation of the various controller-algorithm input features (accessible via a set of utility functions). It also stores the indexes into the pill and power pill bit vectors, in the case that the node has an associated pill or power-pill.

B. Controller Interfaces

The interfaces that need to be implemented to make custom controllers work with the game engine have been kept as simple as possible to eliminate any overhead associated with coding controllers for the competition. Controllers are supplied with a copy of the latest game state at every time step of the game and need to respond with either a single action (Ms Pac-Man controller) or a set of actions (Ghost controllers). The interface for the Ms Pac-Man agent has a single method which takes the game state as input and returns the desired movement direction:

```
interface MsPacManController
  int getAction(GameStateInterface g)
```

The controller for the ghost team is almost identical to the controller above, except that it returns an array of integers, one for each ghost:

```
interface GhostsController
  int[] getActions(GameStateInterface g)
```

Finally, the game-specific information regarding each agent is encapsulated in *MsPacManState* and *GhostState* respectively. These data structure contain information regarding the position of the agent(s), their current direction(s), and, for instance, points scored (Ms Pac-Man) or whether they are edible (ghosts). These data structures are accessible via the game state such that any controller has thus full access to the information regarding all agents at any moment in time.

VI. Sample Controllers

In this section we introduce some very simple sample controllers that are included with the source code to illustrate how the code is to be used. All controllers presented below are reactive and hence are able to respond very quickly to a new game state. As mentioned previously, more advanced controllers are able to determine higher quality moves, yet need to do so in a timely fashion also, as otherwise the game state will have changed too much by the time the move is implemented. We expect participants to strike a reasonable balance between these two requirements. It should be remembered that ghosts are not allowed to reverse direction, so the only choice of direction is to be made at junctions. Despite this, the ghost team controller is polled every game tick and hence can update its planning ready to respond immediately when a decision is actually required.

TABLE II
Scores obtained by the sample controllers playing each other, with a time-limit of 2000 game ticks.

	Ghost teams	
Ms Pac-Man	Random	Legacy
Random	1194	194
RandomNonReverse	2623	1853
NearestPill	4557	3703

We tested all controllers against one another to illustrate the quantitative differences between them. The scores (each averaged over 1000 games) are shown in Table II. It is evident that purely random decision-making may be improved upon very quickly, both in the case of Ms Pac-Man controllers as well as ghost team controllers.

A. Ms Pac-Man

The *Random* controller makes a uniformly random choice of direction at every time step. In case a chosen direction is illegal, the game will repeat the previous move:

```
int getAction(GameStateInterface g)
  return rnd.nextInt(NUM_MOVES)
```

Clearly, a completely random behaviour ignores both the location of the agent, the remaining pills as well as the ghosts. A very simple modification to improve on the above controller is to prevent Ms Pac-Man from reversing (*RandomNonReverse*). This performs slightly better as Ms Pac-Man spends less time going pointlessly back and forth: the controller creates an array list of possible next nodes, which are all the adjacent nodes not including the previous node the agent was at, then selects one of these at random. Finally, the third controller (*NearestPill*) takes the pills into account and heads for the nearest pill (ignoring power pills). However, it should be noted that Ms Pac-Man does so regardless of whether there are any ghosts in the way!

B. Ghost Team

The *Random* ghost team chooses a random direction for each ghost every time the action method is called. The game engine enforces the no-reverse rule, so these random choices are only effective at junctions. The corresponding controller is specified as follows:

```
public int[] getActions(GameStateInterface g)
  for (int i=0; i<dirs.length; i++)
    dirs[i]=rnd.nextInt(NUM_MOVES)
  return dirs
```

The second ghost team controller included with the code is the Legacy Team [11]: each ghost in this team is based on a node value controller, selecting the adjacent node which minimises the distance to Ms Pac-Man according to some distance measure (except for Sue who chooses randomly). These distance measures are:

- Blinky: Shortest path distance
- Inky: Euclidean distance
- Pinky: Manhattan distance

VII. Game Protocols

This section describes the protocols involved in running games between Ms Pac-Man and a ghost team, covering the fundamental requirements, the notion of player and game-engine identities, and the scoring mechanism.

A. Fundamental Requirements

The central requirement of the evaluation system for this competition is that we can conduct a fair evaluation of a set of agents versus a set of ghost teams, and produce a ranked list of the performance of each one. This can be decomposed into evaluating a single Ms Pac-Man controller against a single ghost team, and involves the integration of three entities: the game engine, the ghost team, and Ms Pac-Man. Since our interpretation of the game involves a random element (the ghost team direction reversals), even a game between deterministic players will typically produce a different score each time. Therefore, the evaluation should be based on a number of games to reduce the effects of noise.

Ms Pac-Man is a simultaneous move game, and each player has a short time to respond with a move for each game tick. If a player fails to respond in a specified time (40ms for our competition since the game runs at 25fps) then the game proceeds to the next tick with some default action for that player. Hence, the entities involved in the game must interact asynchronously. This can be done by using multi-threading within a single process, or by using multiple processes. To reduce the risk of interference, we prefer the multiple processes model for competition purposes, though this imposes some inter-process communication (IPC) overhead and is therefore less efficient than the single process model. Whether this matters in practice depends on the communication details, and on the thinking time needed by each agent. Currently we use UDP sockets for the IPC: these work well when run on the same machine, or on a local area network. Whether they can be used over Internet depends on the firewall policy of the institutions involved: the technology itself works well, though the delays may put remote agents at a disadvantage. Typical Internet round-trip delay times (as measured in our lab using the *ping* utility) between Europe and the USA for example may be between 120ms and 180ms. Such delays may cause a slight disadvantage, but are not insurmountable.

B. Player and Opponent Identities

Each player chooses a name, but ideally we need to match the name to a specific instance of the player. Most players have parameters that can be tuned to lead to different standards of play, and may even be adapted to a particular player. Therefore, in order to calculate the true identity of a player we require that a JAR file be submitted with all the classes necessary to run the player. A digital signature (MD5 or SHA checksum) is calculated, and used as an identity key for this player. Due the the ultra-simple Ms Pac-Man and Ghost Team interface design, the players involved currently do not know the identity of their opponent and do not know the previous history of a particular game: the assumption is that they are essentially Markov agents, and all that is needed to be known is encapsulated in the current game state. These assumptions could be easily relaxed by passing the name of the opponent to each getAction request, and also by establishing a unique identity for each game played.

C. Game Engine Identity

Over time the software may support different variations of the game. These could include new mazes, changes to some of the rules, or different parameter settings such as the ghost-reversal probability or the edible time allowed for a particular level. Therefore, when recording the outcome of a game, it is important to note all the details of the game engine. The way we propose to do this is to add a link to the JAR file that incorporates the game. All details pertaining to each game (i.e., the moves made by all players) will be recorded for every game to allow games to be replayed at later stages. While it would be possible to specify many settings in an external parameter file (perhaps in XML format), the option we prefer is to specify it directly in the code. We do this in a special *Constants* class. This saves having to specify a working directory from which to load the parameters (which could be problematic when using a remote class loader, for example).

D. Scoring

Based on past experience of the typical number of entries for CEC competitions, we plan to run a round-robin type league adapted to the bipartite nature of the players involved (i.e., each player is either a Ms Pac-Man or a ghost team). Two leagues will be produced: one for Ms Pac-Man controllers and one for the ghost team controllers. Each Ms Pac-Man controller will play each ghost team the same number of times, and their total scores recorded over all league games played. The Ms Pac-Man agent league is then sorted in order of highest score first, while the Ghost Team league is sorted in order of lowest score first (i.e., the strongest ghost teams are those that keep the Ms Pac-Man agents scores down to a minimum). In case one type of controller is over-represented, the example controllers might be taken into consideration at this stage.

VIII. Conclusions

Games are an ideal test bed for developing novel techniques in computational intelligence and game competitions form an integral part in driving forward the quality of computer controlled players. In this paper we have introduced the Ms Pac-Man versus Ghost Team-competition where participants, for the first time, may contribute controllers for either Ms Pac-Man or the ghost team. The Java implementation of the game, which allows controllers to interact directly with the game engine, attempts to resemble the original game as closely as possible, with only few alterations, chosen to simplify the game. There are many factors that require consideration when designing a competition and we have discussed some of the problems we encountered, alongside the solutions we chose to implement.

Game competitions provide an important service in benchmarking the strength of various approaches to developing intelligent systems and agents. In developing the infrastructure for this competition we have also been mindful of providing other services to researchers. For instance, the way that each player should respond to any game state allows a micro-analysis of player behaviours. For example, a particular game state can be sent to many different players to see how each responds. This can also be used to test players for their ability to solve particular "Pac-Man puzzles", i.e. tricky situations that have interesting solutions, similar to chess puzzles. The core game engine is efficient, and depending on the complexity of the player algorithms may run up to several hundred games per second. This enables rapid experimentation and makes the engine a good platform for developing tree-search algorithms, and we already have a strong Monte Carlo Tree Search player that utilises this [19].

There is plenty of scope for future work. In particular, as controllers improve in performance, it would be desirable to evaluate the controllers against human players and to judge their performance not only based on the game's score, but also based on the enjoyment experienced by the player. This latter aspect, mentioned at the beginning of the paper, is something of great value of the game's industry and we hope to include this aspect in future competitions. One interesting way to incorporate this would be to publish a version of the game on the Web as a Java Applet (since the code is already in Java) and allow human players to choose the ghost team to play against. Given a sufficient number of players playing a sufficient number of games it should be possible to evolve ghost teams that are particularly fun to play against.

Acknowledgments

This work was supported by EPSRC grant EP/H048588/1 entitled: "UCT for Games and Beyond".

References

[1] J. Ahlquist and J. Novak. *Game Development Essentials: Game Artificial Intelligence*. Thomson/Delmar Learning, 2008.

[2] A. M. Alhejali and S. Lucas. Evolving diverse Ms. Pac-Man playing agents using genetic programming. In *IEEE Symposium on Computational Intelligence and Games*, pages 53–60. IEEE Press, 2009.

[3] N. Bell, X. Fang, R. Hughes, G. Kendall, E. O'Reilly, and S. Qiu. Ghost direction detection and other innovations for Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games*, pages 465–472. IEEE Press, 2010.

[4] P. Burrow and S. Lucas. Evolution versus temporal difference learning for learning to play Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games*, pages 53–60. IEEE Press, 2009.

[5] M. Gallagher and M. Ledwich. Evolving Pac-Man Players: Can we learn from raw input? In *IEEE Symposium on Computational Intelligence and Games*. IEEE Press, 2007.

[6] M. Gallagher and A. Ryan. Learning to play Pac-Man: An evolutionary, rule-based approach. In *IEEE Symposium on Computational Intelligence and Games*, pages 2462–2469. IEEE Press, 2003.

[7] T. Haynes and S. Sen. Evolving behavioral strategies in predators and prey. *Adaption and Learning in Multi-Agent Systems*, pages 113–126, 1996.

[8] T. Haynes, S. Sen, D. Schoenefeld, and R. Wainwright. Evolving a team. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 23–30, 1995.

[9] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.

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

[11] S. M. Lucas. Evolving a neural network location evaluator to play Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games*, pages 203–210. IEEE Press, 2005.

[12] S. Luke and L. Spector. Evolving teamwork and coordination with genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 150–156. MIT Press, 1996.

[13] S. Matsumoto, N. Hirosue, K. Itonaga, N. Ueno, and H. Ishii. Monte-Carlo Tree Search for a reentrant scheduling problem. In *40th International Conference on Computers and Industrial Engineering*, 2010.

[14] K. Oh and S. Cho. A hybrid method of Dijkstra algorithm and evolutionary neural network for optimal Ms. Pac-Man agent. In *Second World Congress on Nature and Biologically Inspired Computing*, pages 239–243, 2010.

[15] P. Rajagopalan, A. Rawal, and R. Miikkulainen. Emergence of competitive and cooperative behavior using coevolution. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 1073–1074. ACM, 2010.

[16] A. Rawal, P. Rajagopalan, and R. Miikkulainen. Constructing competitive and cooperative agent behavior using coevolution. In *IEEE Symposium on Computational Intelligence and Games*, pages 107–114. IEEE Press, 2010.

[17] D. Robles and S. Lucas. A simple tree search method for playing Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games*, pages 249–255. IEEE Press, 2009.

[18] J. Rosca. Generality versus size in genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 381–387. MIT Press, 1996.

[19] S. Samothrakis, D. Robles, and S. Lucas. Fast approximate max-n Monte-Carlo Tree Search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 2011.

[20] S. E. Siwek. Video games in the 21st century: The 2010 report. Technical report, Entertainment Software Association, 2010.

[21] I. Szita and A. Lorincz. Learning to play using low-complexity rule-based policies. *Journal of Artificial Intelligence Research*, 30(1):659–684, 2007.

[22] N. Wirth and M. Gallagher. An influence map model for playing Ms. Pac-Man. In *IEEE Symposium On Computational Intelligence and Games*, pages 228 – 233. IEEE Press, 2008.

[23] M. Wittkamp, L. Barone, and P. Hingston. Using NEAT for continuous adaptation and teamwork formation in PacMan. In *IEEE Symposium on Computational Intelligence and Games*, pages 234–242. IEEE Press, 2008.

[24] G. Yannakakis and J. Hallam. Evolving opponents for interesting interactive computer games. *From Animals to Animats*, 8:499–508, 2004.

[25] G. Yannakakis and J. Hallam. A generic approach for generating interesting interactive pac-man opponents. In *IEEE Symposium on Computational Intelligence and Games*, pages 94–101. IEEE Press, 2005.

[26] G. Yannakakis and J. Hallam. A scheme for creating digital entertainment with substance. *Reasoning, Representation, and Learning in Computer Games*, page 119, 2005.

[27] G. Yannakakis and J. Hallam. Towards optimizing entertainment in computer games. *Applied Artificial Intelligence*, 21(10):933–972, 2007.