Ms. Pac-Man Versus Ghost Team CIG 2016 Competition

Piers R. Williams*, Diego Perez-Liebana† and Simon M. Lucas‡
School of Computer Science and Electronic Engineering,
University of Essex, Colchester
CO4 3SQ, UK
Email: {pwillic*, dperez†, sml‡}@essex.ac.uk

***Abstract*—This paper introduces the revival of the popular Ms. Pac-Man Versus Ghost Team competition. We present an updated game engine with Partial Observability constraints, a new Multi-Agent Systems approach to developing Ghost agents, and several sample controllers to ease the development of entries. A restricted communication protocol is provided for the Ghosts, providing a more challenging environment than before. The competition will debut at the IEEE Computational Intelligence and Games Conference 2016. Some preliminary results showing the effects of Partial Observability and the benefits of simple communication are also presented.**

I. Introduction

Ms. Pac-Man is an arcade game that was immensely popular when released in 1982. An improvement on the original Pac-Man game; Ms. Pac-Man added better graphics, additional mazes and new Artificial Intelligence (AI) behaviour for the ghosts. The primary difference that interests academics and researchers is the ghost AI. In Pac-Man the ghosts behaved in a deterministic manner. Ms. Pac-Man added a semi-random element to the ghost behaviours making them non deterministic. This non determinism vastly increased the challenge in creating an effective agent for Ms. Pac-Man.

Ms. Pac-Man has been the focus of two previous competitions. The Ms. Pac-Man screen capture competition [15] which periodically provided the agents with a pixel map of the game and requested the direction of travel. This competition only allowed the entrants to submit agents for the Ms. Pac-Man character. The second Ms. Pac-Man competition was the Ms. Pac-Man Vs Ghost Team competition [22] which was was based on a simulator that mimicked the original game reasonably closely. Entrants had to submit a controller for either the Ms. Pac-Man agent or the ghost team.

This new competition adds Partial Observability (PO) to Ms. Pac-Man. PO greatly increases the challenge in creating good AI controllers. Limited information about the ghosts makes it more difficult for Ms. Pac-Man to plan effectively. Limited information about Ms. Pac-Man forces the ghosts to search and communicate effectively in order to trap Ms. Pac-Man and capture her.

Computational Intelligence (CI) has a long history of using competitions to galvanise research in game agent development. These competitions typically focus on trying to develop the strongest AI for a particular scenario although some exceptions such as the BotPrize [11] competition that focuses on developing Unreal Tournament agents that are human-like. There are many competitions currently active in the area of games. The Starcraft competition [17] runs on the original *Starcraft: Brood War* (Blizzard Entertainment, 1998). Starcraft is a complex Real-Time Strategy (RTS) game with thousands of potential actions at each time step. Starcraft also features PO, greatly complicating the task of writing strong AI. The General Video Game Artificial Intelligence (GVGAI) competition [19] runs a custom game engine that emulates a wide variety of games, many of which are based on old classic arcade games. The Geometry Friends competition [20] features a co-operative track for two heterogeneous agents to solve mazes, a similar task to the ghost control of Ms. Pac-Man.

Previous competitions have been organised that focused on games or scenarios with PO. An early example is the classic Iterative Prisoner Dilemma (IPD) [12], a game featuring a small amount of PO in the form of the simultaneous actions of the two prisoners.

Section II contains a review of the research into the domain of Ms. Pac-Man. Section III contains a description of the alterations to the problem domain since the previous competition. Section IV concludes the paper and describes some possible future work for the competition.

II. Recent Research

A large amount of research has been put into both Ms. Pac-Man and the ghost teams, which is covered in depth in this section.

A. PacMan AI

Gallagher and Ledwich [8] investigated a simplified version of the game including for the majority of their experiments using only a single near deterministic ghost and no power pills.

Lucas [14] explored using a simple Evolutionary Algorithm (EA) in $(N + N)$ form with N = 1 or 10. The EA was used to train the weights for a Neural Network.

Robles and Lucas [21] investigate writing a simple tree search method for writing an agent to play Ms. Pac-Man. This was performed on the actual game using screen capture and a simulator. The tree was formed as every possible path through the maze (depth limit 10) with information in the nodes about ghosts and pills encoded. A simulator of the game was used to evaluate the state of the game in future ticks. The simulator was not identical to the game actually played, leading to some

possible errors in judgement. The authors tried a few heuristics and found that some performed better than others.

Burrow and Lucas [3] compared two different approaches to learning to play the game of Ms. Pac-Man. The paper uses a Java implementation of the game Ms. Pac-Man that allowed easy integration of existing machine learning implementations. The two techniques used were Temporal Difference Learning (TDL) and EA. These techniques were used to train a Multi-Layer Perceptron (MLP) that was then evaluated within the game. The EA was subsequently shown to be superior to TDL.

Handa and Isozaki [10] used Fuzzy logic tuned by a 1+1 EA. The rules were tuned with the EA and consisted of a series of predefined rules about avoidance and chasing as well as pill collecting.

Wirth and Gallagher [27] used Influence Maps to drive a Ms. Pac-Man agent. Positive influence was exerted by pills and edible ghosts, whilst ghosts exerted a negative influence upon the map. The map is then checked in the four cardinal directions that Ms. Pac-Man can move in and the maximum influence is chosen.

Alhejali and Lucas [1] [2] studied the use of Genetic Programming (GP) for evolving heuristics to control Ms. Pac-Man. Some care was needed to prevent the agent from focusing too much on ghost eating instead of pill clearing by using multiple mazes.

Samothrakis et al [24] used a 5 player max^n tree with limited tree search depth. The paper experimented with both Monte-Carlo Tree Search (MCTS) for Ms. Pac-Man and for the Ghosts. Schrum and Miikulainen [25] investigated the use of modular neural networks to control Ms. Pac-Man. The agent was developed for the same simulator as used in the Ms. Pac-Man Versus Ghost Team competition.

Flensbak and Yannakakis [5] describe their solution to the Ms. Pac-Man competition of WCCI 2008. This controller was a largely hand coded agent based around pill hunting and ghost avoidance as its primary tactics. The agent avoids ghosts within a 4x4 grid around Ms. PacMan and then collects pills. With this approach, less time is spent in danger from the ghosts before the maze is reset.

Pepels et al [18] describe their work in creating an entrant to the Pac-Man Versus Ghost Team competition (WCCI'12 and CIG'12). A MCTS agent is described in detail containing a number of enhancements and alterations designed to improve performance specifically in Ms. Pac-Man. Emilio et al [4] worked with Ant Colony Optimisation (ACO) to design an agent for Ms. Pac-Man. Two objectives are chosen to drive the agent. The first is to maximise pill collecting. The second is to minimise being eaten by ghosts. This leads to two types of ants used in the system, the *collector ants* maximising pill collecting and the *explorer ants* minimise death.

Foderaro et al [7] [6] used a tree search technique after abstracting the maze into a connected graph of cells.

B. Ghost Control

Nguyen and Thawonmas [16] present their agent that was entered into the CEC 2011 Ms. Pac-Man vs Ghost Team Competition, subsequently winning. The agents used for this controller were to control Pinky, Sue and Inky with MCTS whilst using a completely rule based approach for Blinky.

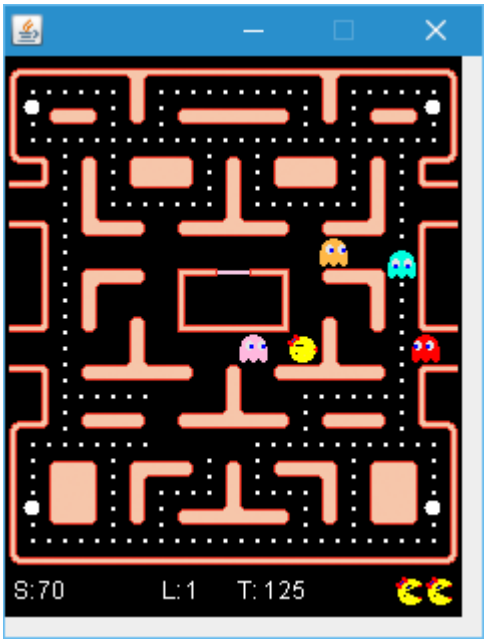

Figure 1. A view of the basic Ms. Pac-Man game

Figure 2. The various characters of the game, Left to Right: Blinky, Inky, Pinky, Sue and Ms. Pac-Man

Wittkamp et al [28] investigate using an online learning technique - Neuro-Evolution Through Augmenting Topologies (NEAT) - to evolve the controllers for the ghosts team. Each ghost evolves separately but shares the score of the team.

Liberatore et al [13] look into the use of Swarm Intelligence (SI) to control the ghost team.

III. The Competition

A. The game

The game that we have based the competition on is the Ms. Pac-Man arcade game. This game consists of 5 agents, a single Ms. Pac-Man and 4 Ghost agents. The world is a maze environment, with peachy coloured walls that are non traversable. There is a ghost lair in the center, where the ghosts start and also respawn after being eaten. Pills are placed in the corridors for Ms. Pac-Man to collect as well as larger Power Pills that allow Ms. Pac-Man to consume the ghosts and score additional points. A view of the game is shown in Figure 1. The various characters in the game are shown in Figure 2. Eating a pill earns Ms. Pac-Man 10 points and eating ghosts earn 200 points for the first ghost but doubling each time up to 1600 points for the fourth ghost. The maximum points s for a maze where n is number of pills in the maze is $s = 10n + 4 \times (200 + 400 + 800 + 1600)$.

B. Partial Observability

PO is the impairment of the ability of an agent to completely observe the world that it is situated within. PO in Ms. Pac-Man can vary in its implementation. We consider some simple methods that could be used, before explaining why we chose the final implementation. First we consider the approach taken in another paper, followed by some simple methods.

1) pocman: PO has been applied to the game of Ms. Pac-Man previously by Silver and Veness [26]. This work covered a small number of domains, one of which was *pocman* - a PO Ms. Pac-Man clone.

In *pocman*, the main agent had to navigate a 17x19 maze and eat the food pellets randomly distributed across the maze. Four ghosts roamed the maze with a simple strategy controlling them. The 4 power pills are also present, allowing *pocman* to eat the ghosts upon contact for 15 steps instead of being eaten in the usual manner.

The ghosts operate randomly unless they are within Manhattan distance 5 of Pacman in which case they chase or evade based on if he is under the effect of a power pill. This implies that the simulation does not support the usual method of ghosts respawning as non-edible before the edible time is up and could be eaten again. This is a major difference to the game that would require some serious modification to the framework in order to replicate. An alternative to replicating this is to make the minor alteration to the ghost AI that it runs away if it is edible and attacks Pacman if it is not edible.

The pocman agent receives a reward to his score each tick from Table I. For example, if the agent moves across an empty square it will receive a reward of -1. This should help force agents to try to finish levels as quickly as possible.

Table I: Table of rewards for *pocman*

Reason	**Reward**
Eating Pellet	+10
Eating Ghost	+25
Dying	-100
None of above	-1

pocman has a particular type of observability in which he is sent 10 observation bits. The first four refer to each cardinal direction and are high if a ghost is present in that directions Line-of-Sight (LOS). The fifth observation bit tells him if he can hear a ghost which occurs when at least one ghost is within Manhattan distance of 2. Four more observation bits refer to the presence of a wall at distance 1 in each of the cardinal directions. The final observation bit is set to high if he can smell food which occurs when a pill is adjacent or diagonally adjacent to him.

2) Line-of-Sight: LOS is where the agents can see in straight lines up to a limit unless there is an obstacle in the way. Obstacles are considered to be the walls in the maze. Ghosts and pills don't count as obstacles. This applies to both Ms. Pac-Man and the Ghosts and means that they can see both forwards, backwards and sideways. This method is simple to implement as well as fairly realistic based on how light travels. Agents cannot see around corners just like real people. This is similar to the standard first person view although we allow full backwards, left and right sight as well.

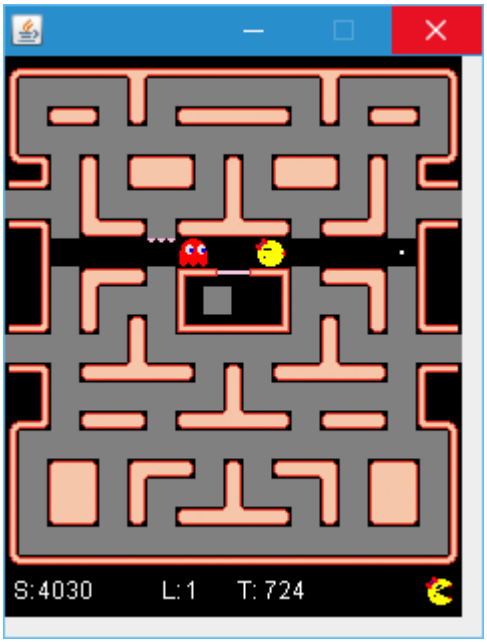

Figure 3. A view of the basic Ms. Pac-Man game with Partial Observability enabled

3) Forward Facing Line-of-Sight: This is an additional restriction on LOS where the agent can only observe in the direction they are currently travelling.

4) Radius based Partial Observability: Radius based PO is a simple technique where we consider anything within a distance d from the agent is considered visible. This technique provides a circular vision when Euclidean Distance is used, and a diamond shaped vision when Manhattan Distance is use. This allows agents to view other agents that are around corners or behind walls. This is not particularly realistic but does provide more information to the agent than LOS.

5) Partial Observability Implementation: The method of PO that has been implemented in Ms. Pac-Man is based on LOS. This we felt was the most realistic without being overly restrictive. The game supports a range limit to the sight - allowing some level of customisation, although at present it is larger than the longest corridor. The view generated by this restriction for Ms. Pac-Man is shown in Figure 3.

With the PO constraints, the ghost entrants must now submit a controller for each ghost. These controllers will be given a 40ms shared time budget, equal to the original competition. The ghosts will be called sequentially in order: (Blinky, Pinky, Inky and Sue). This will allow the flexibility to adjust how much time is spent on each ghost in each tick. This flexibility is useful due to the game rules forcing ghosts to have no actual decision ability when not at a junction in the maze. Locking each ghost into 10ms each would be potentially wasteful for ghosts in tunnels and doesn't leave as much time free for ghosts at junctions.

C. Messaging

Communication is the cornerstone of teamwork and vital to the creation of co-operative agents. In the competition, the communication will be heavily controlled by the game in order to force agents to share information rather than attempt to control the actions of each other. The communication component is composed of two main parts - the messenger and the message. The messages allowed are presented in Table II.

The messages allowed will have a large impact on the ability and even potentially the design of the controllers. In early versions of the messaging system there were more messages planned, allowing the controllers to ask other controllers where they were and where they were heading. Logically it was clear that most controllers would ask for that information every tick and would receive a reply every tick (once enough time has passed). It therefore made sense to simply remove the messages asking and allow the controllers to pass on the information spontaneously. Logically the game could simply provide the information - however the effects of information delay would be lost if that were the case.

The data variable is a single integer at present due to the internal structure of the Ms. Pac-Man simulator. Locations are represented as indices of the node graph, with only a single integer required to show a location in the map. If the messaging system is expanded to include more complex messages, then a more complex system can be used. The extension to a more complex data type for messages would allow even harder messages to use instead of the perfect information messages previously discussed. A more difficult version of the game would only allow an AI to transmit that they can see Ms. Pac-Man or that they can only see Ms. Pac-Man in a certain direction. Imperfect information would lead to more possibilities such as triangulation between reports from multiple ghosts being used to improve the data received.

Messages can be either sent to a single recipient or broadcast to all ghosts on the map. The Java interface for the Messages is presented in Listing 1.

Table II: Table of messages allowed in PacMan

Message Type	**Description**
Pacman Seen	A message informing others that PacMan has been seen.
I Am	A message informing others where the sender is currently located.
I Am Heading	A message informing others where the sender is currently heading.

Other potential messages could be allowed to be passed. Some simple user defined messages for example could allow agents to declare the current strategy they are using. Ghosts could be interested in not just declaring their current position but also their state. Once a powerpill is eaten no ghost knows who has been eaten and who hasn't. An edible ghost could travel towards a chasing ghost for protection if it knew more information.

The messenger system will deliver messages at the time specified by a simple formula. The time it takes to deliver a message for the implementation can be calculated using Equation (1). This allows a level of configurability in how quick the messages get delivered. Each message type has its own cost, the δ_m, and the system has both a multiplier to that δ_x and a constant delay applied equally to all messages δ_c. This allows for example all messages to be delivered equally ($\delta_m = 0$).

The messenger system at present makes no charge for delivering its messages. This allows AI agents to use as many messages as they wish. Introducing the notion of cost to a message would force the algorithms to become more careful with messages and decide whether it is worth sending the message at all. This would increase the level of difficulty for the AI agents as effective and thrifty strategies for messaging would need implementing.

The Java interface for this system is in Listing 2.

Table III: Explanation of terms in Equation (1)

t_d	The tick a message will be delivered.
t_a	The tick a message arrives in the system.
δ_c	The constant delay added to all messages equally.
δ_m	The individual message delay.
δ_x	The constant delay multiplier applied to message delay.

$$t_d = t_a + \delta_c + (\delta_x \times \delta_m) \tag{1}$$

Listing 1. Message Interface

```
public interface Message {

     //Gets the sender of the message
    public GHOST getSender();

    //Gets the intended recipient of the message
    public GHOST getRecipient();

    //Gets the message type of the message
    public MessageType getType();

    //Gets the data associated with the message
    public int getData();

     //Gets the tick that the message was created
    public int getTick();

     //Contains information about message delays
    public enum MessageType {
        PACMAN_SEEN(2),
        I_AM(1),
        I_AM_HEADING(1);

        private int delay;

        private MessageType(int delay) {
            this.delay = delay;
        }


        public int getDelay() {
            return delay;
        }
    }
}
```

Listing 2. Messenger Interface

```
public interface Messenger {

    //Gets a deep copy of the messenger object
    Messenger copy();

    //Updates the messenger
    void update();

    //Adds a message to the messenger
    //to be delivered as soon as it can be
    void addMessage(Message message);

    /**
     * Get all messages that are due to
     * be delivered to me this tick
     * @param querier The agent doing the querying.
     * @return The messages due (could be empty).
     */
    ArrayList<Message> getMessages(GHOST querier);
}
```

D. Sample Controllers for Ms. Pac-Man vs Ghosts

Having implemented PO for both Ms. Pac-Man and the Ghosts an initial trio of new controllers were also implemented that could function within PO. These controllers required the minimum amount of modification to the original basic controllers from the previous competition. The controllers, along with the original starter controllers, will be described next.

1) StarterPacMan (COP): This is the original basic controller for the previous competition and works only in Complete Observability (CO) environments. This controller follows a very basic algorithm with some simple sequential rules as shown in Algorithm 1. The controller will avoid ghosts that are too close, chase ghosts that are edible or travel to the nearest pill.

Algorithm 1 StarterPacMan basic algorithm

```
function GETMOVE()
    limit ← 20
    nearestGhost ← GETNEARESTCHASINGGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVEAWAYFROM(nearestGhost)
    end if
    nearestGhost ← GETNEARESTEDIBLEGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVETOWARDS(nearestGhost)
    end if
    nearestPill ← GETNEARESTPILL()
    return NEXTMOVETOWARDS(nearestPill)
end function
```

2) StarterGhosts (COG): This is the original basic controller for the previous competition to control the four ghosts. It is a puppet master style algorithm, meaning it is a single block of logic that generated moves for all of the ghosts. The controller follows some basic strategies if a ghost is allowed to make a move as shown in Algorithm 2. The ghosts will run away from Ms. Pac-Man if she is able to eat the ghost, or near a power pill (Potential to eat ghost). If the previous rule doesn't apply then the ghost will 90% of the time chase Ms. Pac-Man and 10% of the time move randomly.

Algorithm 2 StarterGhosts basic algorithm

```
function GETMOVE()
    if GAME.DOESREQUIREACTION() = False then return
NULL end if
    pacman ← GETPACMANINDEX()
    if ISEDIBLE() OR PACMANCLOSETOPPILL() then
        return NEXTMOVEAWAYFROM(pacman)
    end if
    if NEXTFLOAT < 0.9 then
        return NEXTMOVETOWARDS(pacman)
    else
        return NEXTRANDOMMOVE()
    end if
end function
```

3) POPacMan (POP): This is a modification of the StarterPacMan where each strategy is followed if it is possible as shown in Algorithm 3.

Algorithm 3 POPacMan basic algorithm

```
function GETMOVE()
    limit ← 20
    nearestGhost ← GETNEARESTCHASINGGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVEAWAYFROM(nearestGhost)
    end if
    nearestGhost ← GETNEARESTEDIBLEGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVETOWARDS(nearestGhost)
    end if
    nearestPill ← GETNEARESTPILL()
    if nearestPill ≠ NULL then
        return NEXTMOVETOWARDS(nearestPill)
    end if
    return NEXTRANDOMMOVE()
end function
```

Other than modifying the original strategies with guards against null, it was clear that a new default strategy was needed. This is because within the PO game, it was possible to proceed through the previous strategies without returning a move. This new default strategy was to simply return a random move.

4) POGhosts (POG): This is a modification of the StarterGhosts where each strategy is followed if it is possible in the PO case. If there is no information available to the ghost, then the ghost will behave randomly at intersections as shown in Algorithm 4.

5) POCommGhosts (POGC): This is a modification of the POGhosts but attempts to communicate each tick in order to improve its chances. If this ghost can see Ms. Pac-Man then it will send a message to everyone else. If it can't see Ms. Pac-Man then it will check if anybody else has seen it. If someone else has seen Ms. Pac-Man then it pretends it can see Ms. Pac-Man and follows the original POGhosts strategy outlined above. This controller will presumably lose capability as the message delay increases due to the reduced accuracy. The pseudo code for this is shown in Algorithm 5.

The threshold used to determine when to forget Ms. Pac-Man's location needs tuning. Every value from 0 to 200 were put to a test on 4000 games against the COP agent and 33,300 games against the POP agents. The results are

Algorithm 4 POGhosts basic algorithm

```
function GETMOVE()
    if DOESREQUIREACTION() = False then return NULL end if
    pacman ← GETPACMANINDEX()
    if pacman ≠ NULL then
        if ISEDIBLE() OR ISPACMANCLOSETOPOWERPILL() then
            return NEXTMOVEAWAYFROM(pacman)
        end if
        if NEXTFLOAT < 0.9 then
            return NEXTMOVETOWARDS(pacman)
        end if
    else
        return NEXTRANDOMMOVE()
    end if
end function
```

Algorithm 5 POCommGhosts basic algorithm

```
function GETMOVE()
    currentTick ← GETCURRENTTICK()
    if currentTick = 0 || currentTick - tickSeen ≥ TICKTHRESHOLD then
        lastPacmanIndex ← −1
        tickSeen ← −1
    end if
    pacman ← GETPACMANINDEX()
    messenger ← GETMESSENGER()
    if pacman ≠ −1 then
        lastPacmanIndex ← pacman
        tickSeen ← currentTick
        if messenger ≠ NULL then MESSENGER.ADDMESSAGE(...)
        end if
    end if
    if pacman = −1 AND messenger ≠ NULL then
        for message in MESSENGER.GETMESSAGES(ghost) do
            if MESSAGE.GETTYPE() = PACMANSEEN then
                if MESSAGE.GETTICK() > tickSeen then
                    lastPacmanIndex ← MESSAGE.GETDATA()
                    tickSeen ← MESSAGE.GETTICK()
                end if
            end if
        end for
    end if
    if pacmanIndex = −1 then pacmanIndex ← lastPacmanIndex
end if
    if DOESREQUIREACTION() = False then return NULL end if
    pacman ← GETPACMANINDEX()
    if pacman ≠ NULL then
        if ISEDIBLE() OR PACMANCLOSETOPPILL() then
            return NEXTMOVEAWAYFROM(pacman)
        end if
        if NEXTFLOAT < 0.9 then
            return NEXTMOVETOWARDS(pacman)
        end if
    else
        return NEXTRANDOMMOVE()
    end if
end function
```

displayed in Figure 4 and show that the value of 50 is a good value against these two agents. Interestingly the data against the POP algorithm is significantly noisier than COP. This is presumably due to COP being deterministic and POP being non-deterministic.

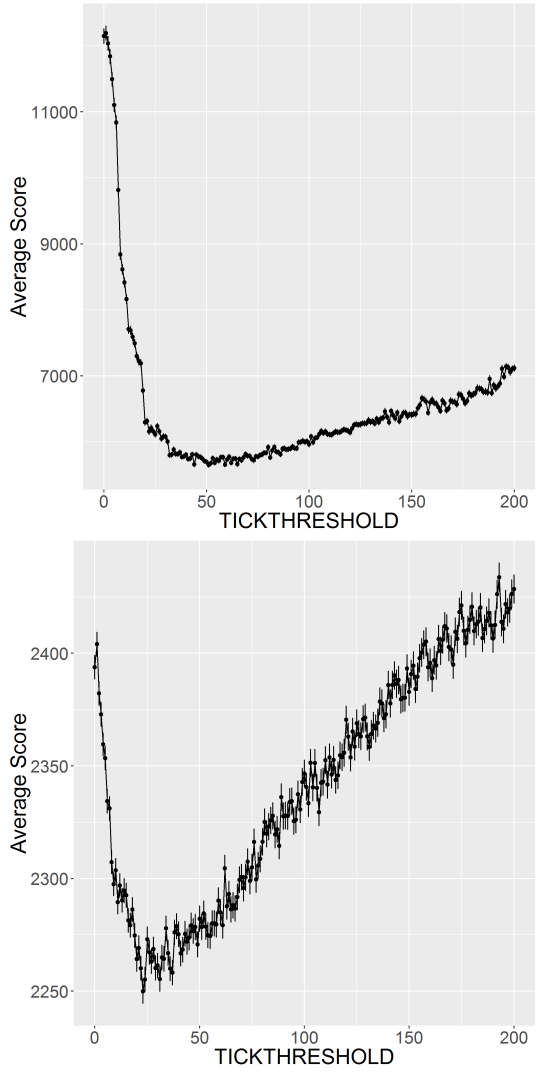

Figure 4. Tuning results of POGC against COP(Top) and POP(Bottom) both with error bars.

E. Sample Controllers Results

These basic controllers may not represent the best agents for the game, but they do provide simple comparisons between them as they rely on the same strategy. The only difference between them is the addition of PO and the effects of this are apparent. Running these controllers over 1000 runs in each combination provides the results displayed in Table IV. It is clear that for the same strategies, PO is a large handicap to the agent. Against COG, adding PO to Ms. Pac-Man caused the score to drop to only 45% of previous performance. Adding communication abilities to the PO Ghosts allowed CO Ms. Pac-Man to achieve only 33.43% of her previous score. This is a huge difference between two very simple algorithms and clearly shows the benefits of communication in this scenario.

F. Competition Tracks

The competition was originally run with two main tracks. The first track allowed participants to submit code to control

Table IV: Table of results after 1000 runs of different controllers

Agents	Mean Score	Std. Error
COP Vs COG	3895.67	48.23
COP Vs POG	17257.24	280.49
COP Vs POGC	5769.30	77.41
POP Vs COG	1753.52	26.97
POP Vs POG	2708.15	37.98
POP Vs POGC	2349.34	30.32

the Ms. Pac-Man character. The second track allowed participants to submit a single class to control the Ghosts.

The revived competition will also feature two tracks. The first track will allow participants to submit code to control Ms. Pac-Man but they will be operating within PO constraints. The second track will allow participants to submit 4 controllers - one for each ghost - that will be operating under PO constraints.

G. Entrant Ranking

While the number of entrants remains low, a round robin tournament will be used for simplicity. If this process begins to take too long, entrants will be assigned scores using the Glicko2 rating algorithm [9] as recommended for competitions [23] similar to this. These will be used to calculate matches in the competition periodically, with these matches updating the scores. The final results will be calculated with a full round robin of the top 10 ghost and top ten Ms. Pac-Man controllers before being announced at IEEE Computational Intelligence and Games Conference (CIG).

IV. Conclusions and Future Work

In this paper we presented a major update to the Ms. Pac-Man Vs Ghost Team competition that will be running at CIG. We presented the PO constraint that has been added to the environment and studied the effect that the ability to communicate has on the ghosts performance when there is incomplete information available. Finally we presented the two tracks: PO ghosts and PO Ms. Pac-Man. The controllers used in this paper were of a very basic nature, and there is a great deal more potential to be realised. This competition aims to explore PO in real time games and communication within PO constraints.

There is a lot of potential work to be done in the future. The current competition still only has 4 mazes for the entrants to play on. Additional mazes can be created and included. The competition has maintained the original balance of one Ms. Pac-Man and 4 ghosts. The competition could be extended to allow for modifications to this balance, with more or less of either type of player.

At present, there is only the one method of sight for observing the environment. The competition could be extended to include additional observations such as hearing or smell. These would be less precise than sight but have the potential to provide information that sight can't.

V. Acknowledgments

This work was funded by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1]

References

[1] Atif M Alhejali and Simon M Lucas. Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming. In *Computational Intelligence (UKCI), 2010 UK Workshop on*, pages 1–6. IEEE, 2010.

[2] Atif M Alhejali and Simon M Lucas. Using Genetic Programming to Evolve Heuristics for a Monte Carlo Tree Search Ms Pac-Man Agent. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[3] Peter Burrow and Simon M Lucas. Evolution Versus Temporal Difference Learning For learning to Play Ms. Pac-Man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 53–60. IEEE, 2009.

[4] Martin Emilio, Martinez Moises, Recio Gustavo, and Saez Yago. PacmAnt: Optimization based on ant colonies applied to developing an agent for Ms. Pac-Man. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 458–464. IEEE, 2010.

[5] Jonas Flensbak and Georgios N Yannakakis. Ms. Pacman AI Controller. 2008.

[6] Greg Foderaro, Ashleigh Swingler, and Silvia Ferrari. A model-based approach to optimizing ms. pac-man game strategies in real time.

[7] Greg Foderaro, Ashleigh Swingler, and Silvia Ferrari. A model-based cell decomposition approach to on-line pursuit-evasion path planning and the video game ms. pac-man. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 281–287. IEEE, 2012.

[8] Marcus Gallagher and Mark Ledwich. Evolving Pac-Man Players: Can We Learn From Raw Input? In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 282–287. IEEE, 2007.

[9] Mark E Glickman. Example of the glicko-2 system. *Boston University*, 2012.

[10] Hisashi Handa and Maiko Isozaki. Evolutionary fuzzy systems for generating better ms. pacman players. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008.(IEEE World Congress on Computational Intelligence). IEEE International Conference on*, pages 2182–2185. IEEE, 2008.

[11] Philip Hingston. A new design for a turing test for bots. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Copenhagen, Denmark, 18-21 August, 2010*, pages 345–350, 2010.

[12] Graham Kendall, Xin Yao, and Siang Yew Chong. *The iterated prisoners' dilemma: 20 years on*. World Scientific Publishing Co., Inc., 2007.

[13] Federico Liberatore, Antonio M Mora, Pedro A Castillo, and Juan Julián Merelo Guervós. Evolving evil: optimizing flocking strategies through genetic algorithms for the ghost team in the game of Ms. Pac-Man. In *Applications of Evolutionary Computation*, pages 313–324. Springer, 2014.

[14] Simon M Lucas. Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. In *CIG*. Citeseer, 2005.

[15] Simon M Lucas. Ms pac-man competition. *ACM SIGEVOlution*, 2(4):37–38, 2007.

[16] Kien Quang Nguyen and Ruck Thawonmas. Applying Monte-Carlo Tree Search To Collaboratively Controlling of a Ghost Team in Ms Pac-Man. In *Games Innovation Conference (IGIC), 2011 IEEE International*, pages 8–11. IEEE, 2011.

[17] Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in starcraft. *IEEE Trans. Comput. Intellig. and AI in Games*, 5(4):293–311, 2013.

[18] Tom Pepels, Mark HM Winands, and Marc Lanctot. Real-time monte carlo tree search in ms pac-man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):245–257, 2014.

[19] Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon Lucas, Adrien Couëtoux, Jeyull Lee, Chong-U Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition.

[20] Rui Prada, Phil Lopes, Joo Catarino, Joo Quitrio, and Francisco S. Melo. The Geometry Friends Game AI Competition. In *CIG2015 - IEEE Conference on Computational Intelligence and Games*, IEEE CIG, pages 431–438, Tainan, Taiwan, August 2015. IEEE Computer Society.

[21] David Robles and Simon M Lucas. A Simple Tree Search Method For Playing Ms. Pac-Man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 249–255. IEEE, 2009.

[22] Philipp Rohlfshagen and Simon M Lucas. Ms Pac-Man versus Ghost Team CEC 2011 Competition. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 70–77. IEEE, 2011.

[23] Spyridon Samothrakis, Diego Perez, Philipp Rohlfshagen, and Simon Lucas. Predicting dominance rankings for score-based games. 2014.

[24] Spyridon Samothrakis, David Robles, and Simon Lucas. Fast Approximate Max-n Monte Carlo Tree Search for Ms Pac-Man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2):142–154, 2011.

[25] Jacob Schrum and Risto Miikkulainen. Discovering multimodal behavior in ms. pac-man through evolution of modular neural networks.

[26] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[27] Nathan Wirth and Marcus Gallagher. An Influence Map Model for Playing Ms. Pac-Man. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 228–233. IEEE, 2008.

[28] Markus Wittkamp, Luigi Barone, and Philip Hingston. Using NEAT for continuous adaptation and teamwork formation in Pacman. 2008.