Ghost Direction Detection and other Innovations for Ms. Pac-Man

Nathaniel Bell, Xinghong Fang, Rory Hughes, Graham Kendall, *Senior Member*, *IEEE*,
Edward O'Reilly, Shenghui Qiu

***Abstract*— Ms. Pac-Man was developed in the 1980s, becoming one of the most popular arcade games of its time. It still has a significant following today and has recently attracted the attention of artificial intelligence researchers, in part, due to the fact that the agent must react in real time in order to navigate its way through the maze.**

This paper forms an entry to the 2010 IEEE Conference on Computational Intelligence and Games Ms. Pac-Man competition, where the objective is to reach the highest score possible without any human intervention. That is, the Pac-Man is under the control of a computer algorithm which must make real time decisions and control the way the Pac-Man moves.

Our Pac-Man algorithm includes detecting the location of the in game objects in relation to the map and creating a grid based graph to represent the game state. Our Pac-Man follows a number of rules, utilising Dijkstra's algorithm and a tree search algorithm.

A further contribution of this paper is effective detection of the ghosts' direction, which we believe has not been done before in the way that we propose.

The world record, for the Ms. Pac-Man competition, is currently help by ICE Pambush 3 with a score of 30,010 (achieved at the 2009 IEEE Congress on Computational Intelligence and Games). Our algorithm consistently achieves a score of over 20,000. The highest score we have recorded is 30,930. We hope that we can replicate these achievements at the 2010 conference where we will present our entry.

I. Introduction

Pac-Man is a popular arcade game developed by Toru Iwatani for the Namco Company in 1981. Since then there have been many different versions of the game. The version used in this paper is Ms. Pac-Man which is very similar to the original.

The original version's objectives were to guide the Pac-Man around the maze collecting all of the pills without being eaten by the ghosts. Once all of the pills in the maze have been eaten, the game will begin again on a new maze which may or may not be the same as the previous map. The more maps the Pac-Man completes, the harder the game becomes. Each map has four power pills in addition to normal pills. When the Pac-Man eats a power pill the ghosts become edible to the Pac-Man for a short period of time. If the Pac-Man eats a ghost during this time he gains extra points, the points gained from eating each ghost grows exponentially. The Pac-Man also gets extra points for eating a fruit which sometimes appears at random on the map. The game ends when the Pac-Man loses three lives. A life is lost after colliding with an inedible ghost.

The Ms. Pac-Man version of the game differs in one key way. The ghosts' behaviour in the original Pac-Man game was deterministic. This meant players could exploit this behaviour by using set routes. In Ms. Pac-Man the ghost's behaviour is non-deterministic which makes the game much more difficult because each ghost behaves differently, making it harder to deploy a strategy against them. The red ghost is the most aggressive, the blue and pink ghosts are less aggressive and the orange ghost is the least aggressive; it behaves in a fairly random way. As more maps are completed, the game becomes harder. Not only do the ghosts speed up, but they also get "smarter". For example, in our experience of playing Ms. Pac-Man, the ghosts seem to cooperate more in order to surround the player from multiple directions, which makes it harder for the player to evade them.

Ms. Pac-Man has attracted significant research interest from the computer science and artificial intelligence community. This is due to the fact that although the game has a relatively simple concept, to eat all of the pills without getting eaten, it is still very difficult as the game is randomised to some extent which means the proposed controller program has to react to the current game situation in real-time.

This paper describes a rule-based system, which involves utilising Dijkstra's shortest path algorithm and a Benefit-Influenced Tree Search algorithm. We are also able to improve performance by a novel method of detecting the direction of movement for each ghost.

In section II we will review related work on Ms. Pac-Man. Section III presents an overview of the current SDK, outlining some of its limitations that we address. In section IV we present our proposed agent design and Section V presents our results, with Section VI concluding the paper and providing ideas for further development.

II. Background

A. Pac-Man

Koza [3] carried out some of the earliest research on Pac-Man, although he used his own implementation of the game. This differed from the real game in that the ghosts all acted in the same manner and their behaviour made the game easier than the real game of Ms. Pac-Man. Koza's approach relied on running a predefined set of rules that the Pac-Man should do, such as "*move along the shortest path towards the nearest pill*".

Wirth and Gallagher [4] used influence mapping in their approach. An influence map describes how desirable places on a map are for the Pac-Man to be. For example, if a location has a lot of pills nearby, then it is a good location. But if a location has a lot of ghosts then it is not such a good location. The idea is to keep moving towards the best possible locations. Using an emulated version of the original Ms. Pac-Man game, this approach achieved very good results.

978-1-4244-6297-1/10/$26.00

Gallagher and Ryan [5] used a simple set of tactics to control their Pac-Man. Their decision making depended on the Pac- Man's position (e.g. Corridor, T-junction) and utilised tactics such as "*If a ghost is less than p away then 'Retreat' else 'Explore'*". Each rule had a weight determined by previous game-play and these were evolved using the Population-Based Incremental learning algorithm [6]. This approach had a number of issues mainly concerning the computational time required for the algorithm to learn.

Robles and Lucas [7] described the use of a simple tree search method, similar to the A* algorithm [8], to control their Pac-Man. Their search creates a tree of all possible routes, judging whether each route is safe or unsafe. For each route a value is calculated by the addition of all the objects on the route, with each object having an associated weight. For example, a pill may be worth two units and an edible ghost five units. The route with the highest value is taken. This approach received excellent results on a simulated version of the Ms. Pac-Man game. Despite achieving high scores, this method is somewhat flawed as the route with the largest value may not be the safest route.

The most recent, and most successful, Ms. Pac-Man agent has been developed by Matsumoto and Ashida [9]. They named their agent ICE Pambush 3. This controller won the 2009 IEEE Symposium on Computational Intelligence and Games (CIG) competition with a world record score of 30,010 points. To achieve this they developed their own screen capture software which extracted the game information more precisely than the original software kit distributed by Lucas [10]. This more precise software kit enabled them to set more detailed rules for controlling their Pac-Man. ICE Pambush 3 [9] combined a set of basic rules with an A* algorithm to decide on the best path. For example: "*IF at least one pill exists AND the nearest ghost is less than eight away THEN move to the nearest pill using the A* algorithm*". Their A* algorithm ignored ghosts if the nearest was over a certain distance away.

Another recent piece of work was carried out by Kelly, Dicken and Levine [13] at the University of Strathclyde. Their work, Strathpac, was another entry to the CIG 2009 competition. The team modified the screen capture component provided with Lucas's SDK to, again, achieve a more accurate description of the map, splitting the map into a series of nodes. While running, the program makes use of two "modes of operation". The mode chosen depends on the current state of the game. If there are edible ghosts present, the controller will hunt the ghost. If not, it will move towards the nearest pill. Theses modes both make use of a breadth first search of the nodes, with movement decisions being made based on predefined priorities of the in game objects. Judging by the controllers ranking in the competition, a breadth first search does not appear to be the optimal approach, in comparison to the A* algorithm. The mode of operation approach also shows some limitations. The controller defines two modes of operation. It might be useful to consider more.

B. Search Methods

We utilise Dijkstra's shortest path algorithm [11] as a way to find the shortest path between two nodes of the map. We also considered using the A* algorithm [8]. The cost of a specific path is determined by the cost of each object along the path. In our case a ghost will have a relatively high positive cost and a pill will have a negative cost. Thus the path with the lowest cost should always be the path with least ghosts and most pills. This meets our goal of evading ghosts and clearing the level. The downside of using A* is that we have to apply the algorithm at every iteration (i.e. after each screen capture) and we have to calculate too far ahead when, in fact, we only need to ascertain the correct direction for the Pac-Man. Moreover, calculating the path using the full map may be neither necessary nor efficient, because in each short time period only a small area around the Pac-Man is actually of interest. Thus the A* algorithm seems to waste a large amount of computational time calculating path information which is not relevant to the short term decisions that need to be made. This could be solved by calculating to a shallower depth. However there is another inherent problem with the A* algorithm. If we were to use a basic implementation of A* we would be combining the cost of pills and ghosts into a single accumulated value. This would potentially lead to situations where the program would choose a more dangerous route because of a higher pill count when we would prefer it to concentrate on evasion. No doubt an effective A* algorithm can be devised, but in this work, we decided not to use it.

III. Software Development Kit (SDK)

In this section, we briefly overview the SDK, highlighting the areas where we have made improvements.

A. Algorithm Overview

After the program has been initialised, a main loop will be started and in each iteration the following actions are performed:

- Get a screen shot of the current game map using the screen capture component.
- Analyse the image and update the game board.
- Determine the best rule to apply according to the current game state.
- Use an appropriate path searching algorithm to determine the best direction to move.
- Output the direction by sending a keyboard action.
- Update the display.

The above series of actions will be performed approximately every 65ms, that is about 15 frames per second. Considering the frame rate of the game, 15 frames per second should be able to keep track of the real-time aspects of the game.

B. Enhanced Screen Capture

It is an essential requirement of the software that the game screen be captured. This is because there is no access to

the internal state of the game. As a result, a screen capture component must be used to convert the real time video into a usable data structure that the system can read. The screen capture component that we use is an extended version of the one provided in Lucas's SDK [10]. The original screen capture takes objects from a screen shot of the game window and updates the information to the game state. The game state will then determine the type of the incoming object and store it in a 2-Dimensional character array. The time to complete image extraction is about 5ms, which is negligible. However, the original screen capture SDK has several aspects which limited the performance of the proposed agent design. Therefore, we made the following enhancements.

1) Ghost Direction: One limitation of the original screen capture SDK provided by Lucas [10] is that it cannot detect the direction of the ghost. As far as we know, no researchers has ever considered the direction of the ghost. In our view it is crucial that the agent's decision making is aware of the direction in which the ghosts are travelling in critical situations such as when multiple ghosts are around the Pac-Man.

Fig. 1. Simple Tree Search Method

In the research of Robles and Lucas [7], a screen shot of the game map has been used to illustrate how their algorithm will choose a safe path (figure 1). In this method, the direction of the ghost is not considered.

Fig. 2. Simulation of the safe path search result of the Simple Tree Search Method [7]

The image in figure 2 illustrates a safe path search result from the Simple Tree Search Method [7]. No safe path will be found since ghost direction is not considered and Pac-Man cannot reach any safe nodes before the ghost. A human player will obviously choose to go downward because the blue ghost is currently moving away from the Pac-Man. If a ghost is moving in the same direction as the Pac-Man, then the agent should decide to follow that ghost if there is no other way to go. This dramatically increases the chances of survival.

In our implementation, a method called "*Ghost Iris Detection*" has been used. As shown in figure 3, the ghosts always look in the direction in which they are moving. The direction of a ghost can therefore be easily determined by examining the position of their eyes. We chose to implement this method, as there is minimal computational cost compared to the more intuitive method of comparing consecutive frames to determine the direction of each ghost. With this method, we do not have to store consecutive frames in order to carry out the comparison as the direction can be calculated from a single static frame.

Fig. 3. These images show how the ghosts look in direction of travel so that we are able to determine the direction of a ghost by examining the pixels of the iris.

2) Teleporter: When a ghost enters the teleporter it cannot be detected because it is *off the screen* (this has also been raised by Robles and Lucas [7]). This leads to two issues. Firstly, a large number of times the Pac-Man is eaten is when ghosts are inside the teleporter, and Pac-Man inadvertently enters the teleporter as it is unaware of the ghost. Secondly, when edible ghosts enter the teleporter the Pac-Man stops following them because it can no longer see them.

This problem is partially solved in our enhanced screen capture by memorising the ghosts' positions for a short period of time when a ghost enters the teleporter and vanishes from the map. Once the ghost comes out the other side of the teleporter, its position is updated again. However, if for some reason it vanishes within the teleporter e.g. it changes state from a normal ghost to an edible ghost whilst in the teleporter, the ghost will be removed after a set period of time. Otherwise Pac-Man will always believe that there is still a ghost in the teleporter when, in fact there is not. This can obviously lead to later problems.

This enhancement ensures that the Pac-Man can always effectively detect the ghosts even if they are in the teleporter while avoiding the problem of having *fake* ghost appearing in the screen capture. However, our current implementation is not able to detect *edible ghosts* in the teleporter since it is not possible to distinguish the four edible ghost which are the same colour. As a consequence, the Pac-Man may stop chasing edible ghosts when they enter the teleporter. We would like to add this functionality to a future release of our algorithm.

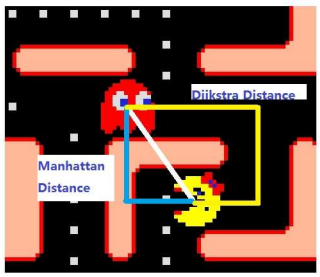

Fig. 4. Distance representation: Straight Line and Manhattan

3) Distance Representation: Initially we used the straight line distance (see white line in figure 4) to measure the distance between the Pac-Man and each ghost. However, this is not realistic as the Pac-Man and ghosts can only move either horizontally or vertically. In a later implementation we used the Manhattan Distance. (the difference in the x-axis plus the difference in the y-axis, shown as the blue line in figure 4). This more accurately reflects the distance of the two objects (Pac-Man and ghost) as it mimics the way in which they move. However, in a game map, there are walls which restrict the movement of the game objects (see figure 4). The Manhattan Distance from the ghost to the Pac-Man is shown as a blue line but the ghost cannot reach the Pac-Man in the distance returned by the Manhattan distance due to the wall.

Our final solution, in order to represent the distance more precisely, uses Dijkstra's algorithm (see the yellow line in figure 4), which guarantees to return the shortest path between two objects. Using this algorithm, we also take the teleporter into account so that we are able to utilise the teleporter when appropriate. Calculating the Dijkstra distance is relatively complex, when compared with both the straight line distance and the Manhattan Distance. To reduce the amount of computation that we have to carry out in real-time, we pre-compute the Dijkstra distances of all possible routes that the Pac-Man can take on the map (we note that a similar technique was also used in [14]). The generation of the distances takes about seven seconds (on a Dell Latitude Laptop with Core 2 Duo, P8600 2.4Ghz CPU) and, once computed, the algorithm can access the distances very quickly (i.e it is just a look up operation of O(1)).

4) Game Board: The game board is an object holding the current game state. In our algorithm, an extended version of the game board is used to represent the current game state in a more realistic and effective way. The map is divided into a 28 $\times$ 31 grid. Each cell is occupied by one or more objects. However, due to the limitation of the screen capture component, only one object is visible in the same cell. That is, each cell has a type representing its contents, these being a pill, a power pill, one of the four ghosts (which we identify separately), an edible ghost (we don't differentiate between the ghosts once they are edible), or the cell is empty.

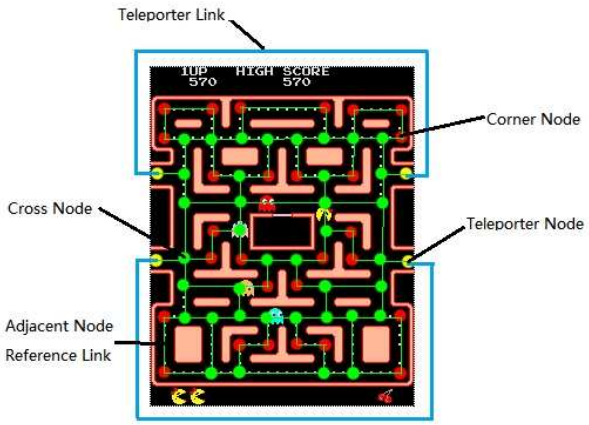

Fig. 5. Graph representation representing connections between node (including the teleporter)

If multiple objects are present in the same cell, the cell type is set to the type which is fully visible on the screen. The grid will be updated regularly by the screen capture software. In addition, information about the position and distance to the nearest pill, power pill, ghost and edible ghost will be stored to facilitate analysis of the current game state.

The game board contains a graph of the map, stored in an adjacent node graph representation. This consists of three types of node, corner node (red), cross node (green) and teleporter node (yellow) (see figure 5). The corner node represents an L-turn on the map while the cross node represents intersections. The teleporter node is a special type of node which represents the *wrap point* located on the two sides of the map. This type of node is used to allow specific operations to be performed when a teleporter node is reached, such as calculating distance through the teleporter or detecting ghosts on the other side of the teleporter.

Each node maintains references to its adjacent node, which are represented by green lines in figure 5. These references will be used in the path searching algorithms. The program should be able to utilise the teleporter for evading or hunting the edible ghost and pills. ICE Pambush 3 [9], which won the 2009 IEEE Symposium on Computational Intelligence and Games, used a concatenated map. This consisted of a complete map in the centre with the right hand side concatenated with the left, and vice versa. In our implementation, a simpler solution is applied by only using a single map. The key point is to make a *Teleporter Node* (a yellow node) on

each teleporter and modify the adjacent node references to connect a pair of teleporter nodes together (blue line).

IV. Agent Design

The Pac-Man agent is one of the core components of the program, which determines the action of the Pac-Man. We use a rule-based method for determining the actions taken by the Pac-Man. Currently there are six hand crafted rules. In each iteration, the updated game board will be passed to the agent object and a pre-defined rule will be triggered according to the distance between different objects. The rules are as follows.

RULE 1: wait for the ghost

```
IF
  distance(nearest_power_pill) <= 5 AND
  distance(nearest_ghost) > 2 AND
  distance(nearest_ghost_power_pill)
  - distance(nearest_power_pill) > 1
THEN
  Wait for the ghost
```

This rule is designed to ambush a ghost when the Pac-Man is near a power pill. If the Pac-Man is with a certain distance of a power pill, a ghost is not about to eat the Pac-Man and a ghost is not approaching the power pill from the other side then wait for a ghost to approach.

Note that

```
distance(nearest_ghost_power_pill)
```

represents the distance between the Pac-Man and the ghost which is closest to the power pill, which is nearest to the Pac-Man. This value is introduced to prevent the ghost from approaching from the other side of the power pill.

RULE 2: eat the power pill

```
IF
  at least one power pill exists AND
  no edible ghost exists AND
  Pac-Man is waiting for ghost (R#1) AND
  distance(nearest_ghost) <= 2 AND
  distance(nearest_ghost_power_pill)
  - distance(nearest_power_pill) <= 1
THEN
  Move to the nearest power pill using
  Dijkstra's algorithm
```

This rule is used when the Pac-Man is already in an ambush state, set by rule one. If a ghost is coming close to the Pac-Man then eat the power pill.

RULE 3: eat the power pill (without waiting)

```
IF
  at least one power pill exists AND
  no edible ghost exists AND
  distance(nearest_ghost) <= 8 AND
  distance(nearest_power_pill) <= 6 AND
  distance(nearest_ghost_power_pill)
  >= distance(nearest_power_pill)
THEN
  Move to the nearest power pill using
  Dijkstra's algorithm
```

This rule is used when the Pac-Man is not in an ambush state and is being chased by a ghost. If there is a power pill nearby the Pac-Man will eat the power pill without waiting.

RULE 4: chase the edible ghost

```
IF
  at least one edible ghost exists AND
  distance(nearest_ghost) >= 9 AND
  distance(nearest_edible) <= 13
THEN
  Move to the edible ghost using the
  Dijkstra's algorithm
```

If there is an edible ghost nearby and there are no non-edible ghosts nearby, move towards it and try to eat it.

RULE 5: eat pills

```
IF
  at least one pills exists AND
  distance(nearest_ghost) >= 10
THEN
  Move to the nearest pill using the
  Dijkstra's algorithm
```

If there are no ghosts nearby, eat the nearest pill.

RULE 6: evade the ghost

```
IF (no rules above match)
THEN
  Evade the ghost using the
  benefit-influenced tree
  searching algorithm
```

If none of the above rules apply, there must be a ghost nearby, so evade the ghost.

The parameters that we use were all set by experimentation and the final values that we used were the best values that we found. These may not (almost certainly will not) be the optimal values and we are aware of some work where these values have been set using an evolutionary approach [4]. We recognise that this is an area that could be fruitful to address in our future research plans.

A. Path Searching Algorithms

Two path searching algorithms have been implemented to deal with specific situations. Dijkstra's algorithm is used for hunting pills and edible ghosts, while Benefit-Influenced Tree Searching is used for more critical situations such as evading ghosts.

1) Dijkstra's Path Searching Algorithm: This is a simple path searching algorithm, which leads the Pac-Man to the goal position via the shortest path possible. The algorithm takes two parameters, the starting position (the Pac-Man's current location) and a goal position where the Pac-Man is trying to move to. As the algorithm is called at each iteration the output does not need to be the entire path to the goal, only the initial move towards that goal. As stated above, the Dijkstra distances have all been pre-computed so that returning the distances is very fast in real-time play. The Pac-Man will then move in the direction that returned the lowest distance to the goal position. If multiple adjacent cells have the same distance to the goal position, the Pac-Man will move to the node in the direction it is already facing.

2) Benefit-Influenced Tree Search Algorithm: When the Pac-Man is searching for a safe path to evade the ghosts we use the benefit-influenced tree search algorithm. Initially a search tree is constructed. The root node is the current position of the Pac-Man. Then we search for the ghosts along the tree path. The costs are calculated according to the following rules:

1) If the ghost is very near to the Pac-Man (for example less than two steps away) then that direction will have a very high cost associated with it in order to prevent the Pac-Man moving in that direction (regardless of which direction the ghost is facing).
2) If the ghost is further than two steps away we check to see whether it is moving toward or away from the Pac-Man. If it is moving away we ignore the ghost.
3) If the ghost is moving towards the Pac-Man we check to see if there are any junctions between the Pac-Man and the ghost that the Pac-Man can reach first. If there are any safe junctions we ignore the cost of the ghost. If the ghost can get there first we add a large cost to that node.

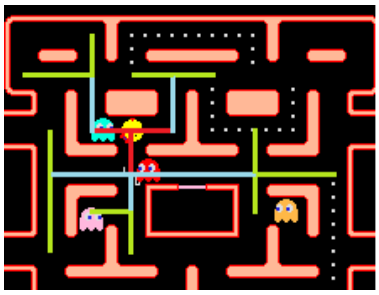

Fig. 6. The different levels while extending routes from the Pac-Man.

To apply these rules to the nodes we start by expanding the child nodes of the current position. These nodes accumulate the costs of all their child nodes by calling the cost calculation method on each child (which in turn do the same on their child nodes). The node expansion will stop when the maximum search level has been reached (this is currently set to 3). Figure 6 shows the search levels which the Pac-Man searches. The image represents the search levels 1, 2 and 3 as red, blue and green lines respectively.

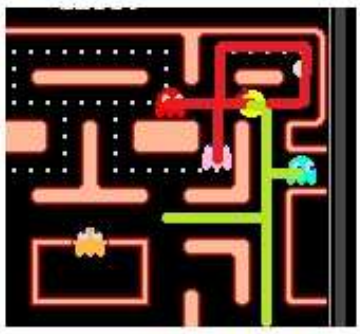

Fig. 7. Safe path search result

When we have searched to the desired depth we move the Pac-Man to the adjacent node with the lowest cost and recalculate. If multiple child nodes have the same low cost then we take into account *benefits*. Instead of maintaining the current direction we take into account the benefit of pills and edible ghosts and pick the route which provides the highest benefit. This means that when evading we are still able to accumulate points. Once this algorithm has been implemented, the problem shown in figure 2 is solved. In our implementation, the ghost direction will be detected. Thus the Pac-Man will ignore the blue ghost (as it is moving away from the Pac-Man) and the path to go downward, which is marked by the green line in figure 7, will be considered safe.

B. Eating the last pills

One issue we observed with the agent is that despite having an opportunity to clear the last pills on the map it would continue to evade the ghosts and avoid eating the remaining pills when an opportunity presented itself. In order to remedy this we implemented a new rule:

- When the pill count for the whole map is less than a predetermined amount we check to see if the benefit value of a route is the same as the remaining pill count.
- If this is the case, then we know that a particular route contains the remaining pills.
- When this happens we add a very large benefit to that route in order to force the agent, when possible, to take it in order to complete the level.
- In these circumstances we ignore the ghosts which could lead to the Pac-Man being eaten.

Our future work will add additional functionality to check whether or not the last pill(s) can safely be eaten as the strategy described above can sometimes lead to problems.

V. Results

The algorithm we use for the controller has been continuously changing since we introduced the general concept of creating a grid of the Ms. Pac-Man map. Although the essential idea of the algorithm has not been changed, there have been key additions and changes which have significantly improved the algorithm.

When we received Lucas's SDK we tested it by running it ten times. In this test session it scored a maximum of 5,410 points and averaged 2,743 points.

After implementing our proposed grid algorithm (noded graph) we carried out another test. Over ten runs it scored a maximum of 17,140 points and averaged 10,766 points. This demonstrates the effectiveness of the grid algorithm.

Our final version, containing all the innovations described in this paper, averages around 18,000. It occasionally (but rarely) scores around 6,000. When this happens it struggles to clear the first level. For most of the runs, though, it scores over 15,000 and gets to at least the third level. The highest score it has recorded is 30,930 points. In doing so it reached the fifth level. This score, if achieved in the competition, would be a new world record, beating the score of Pambush 3 of 30,010 points, in the 2009 CIG competition. Of course, other competitors in the 2010 competition, might perform even better.

In the 2009 competition, Pambush 3 recorded an average of 17,102 points over 10 rounds. We ran a controlled test over ten runs (rather than the empirical evidence we presented above). The maximum score from those ten runs was 24,640, averaging 17,994. Although the highest score was not as high as Pambush 3's at CIG 2009, the average was slightly higher. We believe that this demonstrates that our Ms. Pac-Man controller is operating at a competitive level. Figure 8 shows two different versions of our algorithm, and the original SDK version (i.e. the one downloaded from the competition web site). We show each algorithm's lowest, average and highest scores. This confirms that our most recent algorithm is superior to the previous two versions.

Our current version still has some shortcomings, the main one being that the Pac-Man tends to die in similar ways each time, these being:

- When the ghosts turn from edible to inedible and the Pac-Man is just about to eat it, sometimes the Pac-Man is too close to the ghost so does not have time to escape.
- The Pac-Man sometimes gets trapped in an area of the map where there are little (or no) opportunities to escape so it gets surrounded by ghosts. This tends to happen when the Pac-Man is eating the last remaining pills, after it has eaten all four power pills.

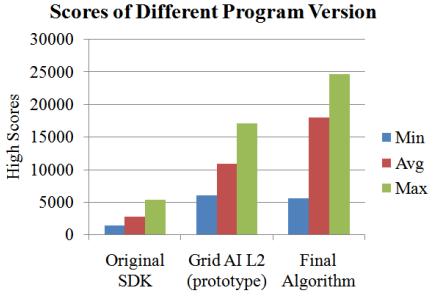

Fig. 8. How our algorithms have improved at each version.

VI. Conclusions

We have described a rule-based system which selects an appropriate strategy for Pac-Man to use. Two path searching algorithms, Dijkstra's Shortest Path and Benefit-Influenced Tree Search algorithms have been used in order to help the Pac-Man decide both the shortest path to a target and the safest path to evade ghosts. A method to detect which direction a ghost is moving has also been introduced. *Iris Detection* uses the ghost's iris to determine which direction the ghost is moving as the ghosts always look in the direction in which they are moving. We do not believe that this method has been implemented before. This has led to more accurate information that leads to a significant improvement to the score.

Several problems in screen capture and game state modelling have also been addressed. The ghost becoming invisible while crossing the teleporter problem has been solved by remembering the ghost's last position for a short term. In addition, using Dijkstra's algorithm has shown benefits.

However, there are still some areas that we would like to address in future work. One issue is the strategy for eating the edible ghost(s). Consuming as many edible ghosts as possible greatly affects the score as consuming a ghost doubles the reward with every successive ghost. Currently our strategy only chases the nearest edible ghost, after eating a power pill, which may not always be the best strategy. A more intelligent plan is likely to be possible, which could lead to a higher chance of consuming all four ghosts. Another issue is the choice of the parameter values in the rule-based system. Currently the parameters used were determined by manual experimentation. We recognise that even better results might be achieved by optimising the parameter, perhaps using an evolutionary approach.

We know that the ghosts have different levels of aggression. It might be beneficial to prioritise the most aggressive ghosts, which might mean a longer life for the Pac-Man.

We would like to analyse why we have only ever got to the fifth level. Perhaps the ghosts are getting smarter? Perhaps we need to change strategies at higher levels? If we are able to develop custom strategies for each level it could lead to more effective agents.

Finally, since eating pills slows down the Pac-Man, it might be a better approach to make the Pac-Man prefer an evasion strategy, when being chased, and consume less pills so that it can move faster.

VII. Acknowledgements

We would like to thank Simon Lucas for supplying the original SDK. Without this development kit it would not have been possible to enter this competition.

We would also like to acknowledge the support of the School of Computer Science at the University of Nottingham, UK. The development of the software reported in this paper was the result of an undergraduate group project. Due to the promising results the School financially supported one of the students to attend the conference and enter the competition.

REFERENCES

[1] Simon Lucas's Ms. Pac-Man webpage, http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html, 30/03/2010.

[2] Ms. Pac-Man online emulator, http://www.webpacman.com, (22/03/2010).

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

[4] N Wirth and M Gallagher, "An Influence Map Model for Playing Ms. Pac-Man," in 2008 IEEE Symposium on Computational Intelligence and Games, pp. 228-233.

[5] M. Gallagher and A. Ryan, "Learning to play Pac-Man: An evolutionary, rule-based approach," in Congress on Evolutionary Computation (CEC), R. S. et. al., Ed., 2003, pp. 2462-2469.

[6] S. Baluja, "Population-Based Incremental Learning: A method for integrating genetic search based function optimization and competitive learning," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-94-163, 1994.

[7] D. Robles and S. Lucas, "A Simple Tree Search Method for Playing Ms. Pac-Man," in IEEE Symposium on Computational Intelligence and Games, 2009, pp. 249-255.

[8] P. Hart, N. Nilsson and B. Raphael, "Correction to A Formal Basis for the Heuristic Determination of Minimal Cost Paths," ACM SIGART Bulletin, issue 37 (December 1972), pp. 28-29.

[9] R. T. Hiroshi Matsumoto, Takashi Ashida, "Ice pambush 3," a 2009 IEEE Symposium on Computational Intelligence and Games competition entry.

[10] S. Lucas, distributed a screen-capture software kit. Retrieved from http://cswww.essex.ac.uk/staff/sml/pacman/pac.zip, (13/11/2009).

[11] Dijkstra, E. W. "A note on two problems in connexion with graphs." Numerische Mathematik, 1959, 1: pp 269-271.

[12] Tickle arcade emulator, created by Alessandro Scotti. Retrieved from http://www.ascotti.org/programming/tickle/tickle.htm, (23/03/2010).

[13] L.Kelly, L.Dicken, J.Levine, "StrathPac - An Automated Player for Ms. Pac-Man", a 2009 IEEE Symposium on Computational Intelligence and Games competition entree.

[14] S.M. Lucas, "Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man", IEEE Symposium on Computational Intelligence and Games, 2005, pp 203-210