Using a Training Camp with Genetic Programming to Evolve Ms Pac-Man Agents

Atif M. Alhejali and Simon M. Lucas
Game Intelligence Group
School of Computer Science and Electronic Engineering, University of Essex, UK
{amalhe,sml}@essex.ac.uk

Abstract — This paper investigates using a *training camp* in conjunction with Genetic Programming in the evolution of Ms Pac-Man playing agents. We measure the amount of effort, time and resources required to run the training camp successfully. The approach is compared with standard GP. The results indicate that better and more stable performance can be achieved using the training camp method at the expense of greater manual effort in the design of the training scenarios. However, in addition to the better results, the training camp also provides more detailed insight into the strengths and weaknesses of each controller.

Keywords: Pac-Man, Genetic Programming, Evolving Controllers, Decomposition learning, Training camp.

I. INTRODUCTION

Games provide an excellent test bed for machine learning techniques due to their complexity, variety and intellectual challenges in addition to the fact that they are easy to implement and test. Games such as Chess, Checkers, Go and Pac-Man, are well-known to Artificial Intelligence (AI) researchers. Common methods for developing AI players include Tree Search, Reinforcement Learning and Genetic Programming (GP). In this paper, the classic arcade game Ms Pac-Man is used as an environment in which to test the ability of the training camp method in GP to create a complete controller that can compete against the controller created with standard GP, using the same function set and under similar conditions. The idea of the training camp is to break the problem down into a set of smaller representative problems that can easily be solved and learned before gathering all the solutions created for these sub-problems to create a complete policy.

With this work we aim to answer several questions. The main question concerns the training camp's ability to create better agents than standard GP. The second question involves the efficiency of the training camp, compared to that of a traditional GP run. Additional questions that will be investigated include the amount of effort the developer must put into each method in order to achieve the best possible solution and to create the training scenarios for the different training camp phases. Beyond the possible improvements in performance, we are also interested in using training scenarios as test-cases to provide a fine-grained analysis of an agent's behaviour and abilities. Even in a relatively simple game such as Ms Pac-Man, there are many distinct behavioural traits which may be unclear from looking at aggregated statistics such as average score or average number of levels cleared. Detailed test-cases may identify escaping behaviours of rare quality, or expose weaknesses where an agent dies unnecessarily.

Also of interest is how best to use the training scenarios. In this paper we take the following approach:

- group the scenarios according to the specific behaviour we are trying to evolve (such as escaping from dangerous situations);
- evolve agents specifically for each type of behaviour;
- use each evolved agent as a function node in a final GP run whose aim is to learn how best to combine the individual behaviours.

Clearly there are many other ways that this could be done, but our initial testing showed this method to work well.

II. THE TRAINING CAMP

For real time applications such as video games, the standard GP approach is to use overall game performance as a measure of fitness (perhaps based on several runs to ameliorate the effects of noise). During a game an agent may face a variety of situations, from the easy to the impossible. When games involve randomness (either from the nature of the game, or from non-deterministic opponents) luck plays a significant role. This is especially true in Ms Pac-Man, where the same agent may score 3,000 on one run and then 30,000 on the next run. A possible solution to this problem is to run each individual several times on the simulator and then calculate the average fitness. However, with a large population, this technique can become expensive, yet the probability of lucky individuals compromising the final findings is not entirely diminished.

The idea of training camps began as a possible solution to this problem. The theory behind the training camp is taken from the way in which young footballers are taught in training camps, where they learn how to control, kick and gain position of the ball, in addition to

many other skills, prior to being allowed to implement all that they have learned in a real game.

In a computer game (or when dealing with any other type of problem), the overall complex strategy required to solve the problem is decomposed into a group of simpler tasks. A set of training scenarios will be created for each task, in which all of the elements in the environment are set to begin in a specific state and position. The scenarios need to be as comprehensive as possible for each task in order to train a skilful agent capable of facing real life situations.

Practically, the work should begin by dividing the main goal into several sub-tasks, which, when combined, should result in the final aim. For each sub-task, a set of scenarios will be created (manually or automatically) so as to cover all possible aspects of this task. These scenarios should be carefully designed to meet two conditions:

1. They should vary in terms of the level of difficulty so that agents do not learn how to solve a very difficult situation and then fail in a simpler one.
2. They should incorporate different environmental elements, as agents may otherwise fail to solve a similar situation merely because a certain environmental element differs from the training scenario.

Designing the scenarios includes determining how long each scenario should run, namely just long enough to complete the task. The last aspect of the scenario is the fitness function, which can differ from one task to another but must be the same in all of the scenarios within the same task.

After designing all of the scenarios, the sub-agent evolving process will begin by running the GP for each task. There is nothing special about these GP tests, other than the fact that the fitness will be calculated according to the fitness function after running each individual in the population on the scenarios designed for this task for the specific time designated for each scenario. The GP system can be run simultaneously on the same or even different machines for different tasks. After finishing the runs, the best evolved individuals will be collected and added to the function set to be used by the final GP run as black boxed sub agents. This final run will produce an agent that can benefit from the specially trained sub-agents as terminals to be called to give the correct answer according to their training as needed.

III. PREVIOUS RESEARCH

Since the training camp idea is based on the idea of problem decomposition, the most relevant paradigm is that of Layered Learning, in which agents learn how to perform a simple task in the first level and then advance in the next level to be able to do another task that involves the previous task or a portion of the previous task joined with the new task in order to create a more complicated controller. This method has proven its ability to create reliable solutions when it is combined with GP. Jackson and Gibbons [1] tested Layered learning against Koza's Automatically Defined Functions (ADFs) [2] and found Layered Learning to outperform ADFs for reasons related to the idea of focusing on a small problem, solely with the required functions, without wasting computational resources on unneeded processes. In games, Layered Learning has proven itself to work well with robot soccer. Bajurnow and Ciesielski [3] made use of layered learning in order to teach players how to score goals. They began by evolving a goal-making technique, in which the robot has the ball and has to learn how to kick it into the goal while avoiding a static goalkeeper. In the next step, the players learn how to find the ball in the pitch and must then apply what was learned in the previous step to score a goal. Gustafson and Hsu [4, 5] used Layered Learning with Multiple-Agent GP, as did Bajurnow, and Ciesielski [3], and they tested it with the Cooperative Robot Soccer Problem. They divided this problem into a set of sub-problems and began by applying GP to one sub-problem. The final generation in this sub-problem was used as the initial generation of the next sub-problem. Their method produced agents that were able to achieve a superior level of fitness in less time.

We now review some results on simulated Pac-Man. Note that many of these results have been obtained on different simulators of the game, and are not directly comparable. Koza [2] used his GP system on a Pac-Man simulator containing the basic components of the original game. With a ghost team that was not very aggressive and moved completely randomly 20% of the time, the final controller was able to achieve 9,420 points out of a possible 18,420. Alhejali and Lucas [6] used a function set inspired by that of Koza on the Ms. Pac-Man game. With more aggressive non-deterministic ghosts, fewer points to be scored in any single maze, and more points to be scored in the four mazes that the agent was able to clear, the agent was able to attain an average score of 16,014 and a maximum score of 44,560 out of a possible 58,160.

Lucas [7] used neural networks to create a reactive agent. His method was based on testing the current node's neighbours and selecting the one with the highest value. This work included a comparison between evolving agents for the deterministic and the non-deterministic versions of the game. The non-deterministic version proved to be much harder but far more interesting. Martin et al [8] created an optimized Ms Pac-Man agent based on Ant colonies and using genetic algorithms. The best produced agent developed using GA and the game simulator were tested in a real environment using a screen capture mode; the agent was able to score a mean of 10,120 points with a maximum of 20,850 points in 20 tries. Bell and others [9] created a Ms Pac-Man agent using Dijkstra's algorithm to select the shortest safe path to a target. They scored an average of 17,990 points and achieved a maximum score of 24,640 on the CIG 2009 simulator. Keunhyun

and Cho [10] also used Dijkstra's algorithm to create a hand-coded set of rules to play Ms. Pac-Man. In their work, they combined the hand-coded rules with evolved Neural Networks and demonstrated that the hybrid technique is a clear improvement.

Many researchers have employed other AI techniques to teach the computer how to play Pac-Man. For example, the Monte-Carlo Tree Search (MCTS) approach has grown in popularity over the last several years and has been tested on Pac-Man more than once. Tong and Sung [11] used MCTS to give Ms Pac-Man the ability to avoid the ghosts in order to enhance the final score. They were able to attain a maximum score of 21,000, and in some cases, the agents were able to reach level 4. Robles and Lucas [12] applied a simple Tree Search method on a Ms. Pac-Man agent. With this method, the agent was able to score an average of 9,630 points on the screen capture original version of the game and over 40,000 points on their simulator. More recently Samothrakis, Robles and Lucas used MCTS to create high performance agents and ghost teams [13].

GP has been popular with other games besides Pac-Man. Langdon and Poli [14] utilised GP to create a Solo Pong player. The best agent resulted from a combination of a hand-coded optimal player and GP-evolved player, and outperformed a human player. Other examples include Benbassat and Sipper [15], Ebner and Tiede [16], and Luke [17], who applied GP to create controllers for Lose-Checkers, TORCS and football, respectively.

IV. THE GAME

The Namco Company first introduced its arcade prey and predator game Pac-Man in 1980. This game was an instant and enduring hit and became one of the most popular arcade games of all time. The following year the Ms Pac-Man variant of the game appeared. This was made more interesting and more difficult due to non-deterministic ghosts, multiple mazes, and moving fruit. Space does not permit a full description of the game to be given here, it is described in more detail in recent publications such as [13]. The simulator used in this paper is the same simulator that was used in the CEC 2011 Pac-Man vs. Ghost team competition[1]. It contains the four mazes from the original game arranged into levels in which the edibility time after a power pill is consumed decreases with every level. All of the tests in this paper have been conducted against the Legacy Ghosts Team under the same conditions.

V. EXPERIMENTAL SETUP

The aim of this work was not only to determine if the training camp is able to achieve better results than standard GP, but also to investigate the extra requirements. The practical work was divided into two parts. The first part entailed creating the function set to be used, which needed to cover all aspects of the game and needed to be proven to work and to reach the required level. The second part involved using the same function set in both standard GP and training camp runs with the aim of ensuring a fair comparison.

A. *The Function Set:*

The function set used in this research is a revised version of the one previously used by the authors [6]. It is divided into two types. The first is comprised of general mathematical and logical operations that help the agent make decisions in the tree according to the information given by the other part. This first type is referred to as "Function", and nodes of this type in our system always have between two and three children in order to direct the tree navigation to one of them. The second type is called "Terminal"; nodes of this type return information about the current state of the game or give orders for the controller to follow. These types are included in three categories:

- Arithmetic:
  - **Functions**: Functions take two children (*arithmetic terminals*) and return the result of basic mathematical operations between them (+, -, ×, ÷).
  - **Terminals**: Terminals return information in numerical form. This information can be categorized into three main types. The first type is the distance between Pac-Man and an object in the maze, such as the nearest food pill, all of the ghosts, whether edible or inedible, and all of the energizers. The second type returns the remaining edible time for each ghost. This is the time in which Pac-Man must eat the ghost before it becomes a predator again. The last type returns a random constant number.
- Logical:
  - **Functions**: Functions take two children from the *logical terminals* and compare them using logical operations (AND, OR) and return the result in the form of a logical value (TRUE, FALSE).
  - **Comparison Operators (functions):** This is another type of logical function that will return either TRUE or FALSE. The difference in this type is that it compares two arithmetic terminals using the comparison operators ($>, \geq, <, \leq, =, \neq$)
  - **Terminals**: These terminals answer certain yes/no questions, such as is the first ghost edible? have all of the energizers been cleared? There are 5 terminals in this category: *IsEdible*, *IsInDanger*, *AreEnergizersCleared*, *IsToEnergizerSafe*, and *IsToEdibleSafe.*
- Action:
  - **Functions**: There is only one function in this type, namely the IF statement (ifte). This function takes a first child that returns a logical value (any logical

[1] http://www.pacman-vs-ghosts.net/

function or terminal), and depending on this value, it will select the second or third child from the same type (function) or from an action terminal.

- **Terminals**: These are the terminals that will direct the agent to its target. The return value is a number that indicates the direction of the next step in the maze (up, down, left, right). There are two terminals for each target in the maze; one of them chooses the next step in the shortest path and the other one offers the next step in a safe path. As all of the distances between the nodes in the maze are pre-calculated at the beginning of the game and stored in a two-dimensional array; it is only a matter of finding the closest neighbour node to the target, using the same table, and then moving Pac-Man to it. Finding a safe path, in contrast, is different. The terminal starts with the shortest path and checks all of the nodes in it for ghosts that could jeopardise this route. If it is safe, then Pac-Man will be directed to the first node in the route; otherwise, the terminal will check the second shortest path and then the next shortest, up to 15 paths. The reason for limiting the choices is the fact that all of the routes in the maze are connected; thus, an unlimited number of paths would cause the terminal to go on in circles, forever searching for the right path without reaching the target. The terminals in this category are either designed to get the agent closer to the target or in the opposite direction away from it.

B. *Standard GP:*

The complete function set is available for GP use. The population size was set to 400 individuals, and run for 100 generations. The mutation probability was 0.3, leaving the remaining 70% of the offspring to be created by crossover. Each individual will run in the simulator for a specific number of times (in this case 5 times) without a time limit and with one life per try. The fitness is the average score of these runs. A list of the top 100 individuals is kept and updated every generation; at the end of the run, these individuals will be tested 100 times each so that the best among them can be chosen. This resampling approach is used to reduce the effects of noise.

C. *The Training Camp:*

As stated above, the training camp involves a set of scenarios to solve where these are grouped into related sub-problems. Several agents will be created using GP to solve the sub-problems. They will be trained on a set of scenarios that cover most of the situations that they may face in the real game. These scenarios are only run for the time required to solve the problem and are then terminated. After each sub-agent reaches a satisfactory level, they will be gathered and used as terminals (action terminals) in the final run in order to create the final product, which is a complete agent that is capable of playing the full game at a competitive level.

Effective Pac-Man playing involves at least three different skills: how to eat the pills effectively, how to escape from the four ghosts and how to chase them and eat them when it is possible. Each one of these skills is composed of several different factors, and the player needs to be prepared to face different situations during the game in order to survive and to achieve a good score. After learning these skills, it is essential to learn when to use each of them.

As mentioned in section II, which described the Training Camp, the most critical part is designing the scenarios to meet the requirement of each task. The three tasks with which we are concerned in this research are clearing the Maze, escaping the ghosts and chasing the ghosts.

Clearing the Maze:

The simplest strategy for clearing the maze is to go after the pills whenever there is no danger in doing so.

The first scenario that can be used to train Pac-Man for this task is a maze that only contains pills (no power pills or ghost). The result of this training was a tree with a size of 1 (*toPill*). Adding more elements to the maze did not have any significant impact. A maze in which power pills were added to the food pills did not change the results and even adding a ghost or more had no impact apart from a significant reduction in the final score with no change in the produced tree. The trees did not change because the fitness value of these scenarios was the final score. Without power pills GP was unable to clear the maze reliably, despite the fact that strong agents are able to do this.

As a result of these findings, no further research has been done in this area, and the *toPill* terminals that use the shortest and safest paths have been used in the final stage as sub-trees.

Escaping the ghosts:

The scenarios for this task need to put Pac-Man into difficult situations where sub-optimal moves lead to a high risk of being eaten. In each scenario, Pac-Man will start in a specific location in the Maze, while each of the ghosts begins from a different location. At least one of the ghosts should be in a position dangerous to Pac-Man.

The first set of scenarios consisted of 10 different situations in which the agent is endangered by the presence of one or more ghosts. Each scenario contained only the power pills in addition to the agents and the four ghosts. The agents were given 50 time steps for each scenario and only one life. The fitness value was the average time that the agent can survive over all of the scenarios. The positions ranged from one or more ghosts chasing Pac-Man in the middle of the maze from one or more directions in order to corner it away from the power pill, with only one way to escape,

to the final method of trapping him in a corner, in which only eating the power pill can save it.

After the first test, a major problem was identified with these scenarios. Because the power pills were used in the maze, all of the agents learned strategies that depended on them. In a real game, the agent would die shortly after eating the last power pill (after the edibility time expired) because it had never trained to escape the ghosts in a power pill-free world. Other problems related to the ghosts themselves and their reversal policy were noted. Each ghost only moved forward; thus, at any point in the game, it was unable to return to the node that it had occupied in the last time step, unless it was reversing. The reversing point is pseudo-random in our simulation of the game with a low probability that all of the ghosts are forced to reverse on themselves, regardless of their current state or direction. This policy caused the entire scenario to collapse at certain points because the ghosts were forced to reverse, giving Pac-Man an easy escape. In the next scenarios, the reversing was disabled. Another problem found after testing the first output was the ghosts' start delay. The first ghost starts at game tick 0, while the second starts after 10 time steps (in this simulator) and the third after 20, while the last ghost makes its first move after 40 time steps have elapsed since the beginning of the game. This delay was eliminated in the next tests because these scenarios should occur in the middle of the game, in which the ghosts are already on the move. To make the environment more challenging, all of the ghosts were forced to make their first move directly towards Pac-Man; thereafter, they could choose their moves according to their original behaviour.

In the next test, two scenarios were produced for each one in the first set, one with power pills in the maze and one without them, in addition to all of the other amendments. Another change made was to the running time, which was increased to 300 time steps because it had been noted that in some cases, the agent did not have enough time to escape completely from the danger zone within only 50 time steps. The agents produced in this test did not perform as well as in the first run because of the poor design of the scenarios. In most of the cases, an agent that was able to escape from the ghosts in one scenario was able to do it in its twin scenario with or without the power pills.

This led to creating a third set of scenarios using the same ideas of Pac-Man being chased or trapped by the ghosts in different places with different numbers of ghosts without the power pills for 20 scenarios and with them for another 10. Pac-Man might be close or far from the pill and was always chased by the ghosts. In some of these scenarios, the agent has no way to escape other than by eating the power pill; in other scenarios, it has the choice of eating it or finding another way. The test time was reduced in this run to 100 time steps. This change was made after noticing that in most of the scenarios, the agent was able to escape the danger zone in around 100 time steps, rendering the remaining time wasteful.

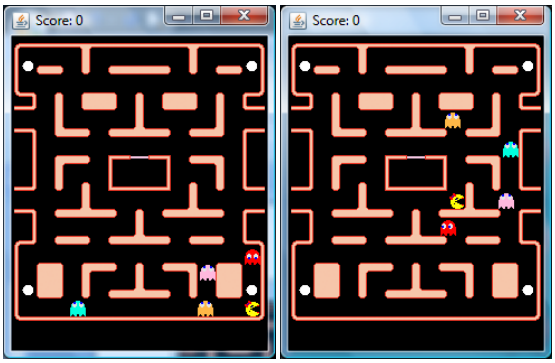

Figure 1: 2 examples of scenarios that have been used to train the agent to escape ghosts

Finally, in order to reduce the effects that the randomness in the ghosts' behaviour has on the result, each individual runs each scenario twice, with the fitness recorded as the average survival time.

Chasing the Ghosts:

Two sets of scenario were created in this section in order to train Pac-Man to learn how to catch as many of the ghosts as possible within the edibility time. In the first set, Pac-Man will start at a point close to the power pill, with the ghosts in different positions and at different distances. This scenario included 310 time steps because the edibility time for the first level is 300 time steps, thus giving the agent 10 time steps in which to eat the pill and then the edibility time in which to eat the ghosts. All of the food pills were removed from the maze so that their portion of the score would not interfere with the final score or affect the produced behaviour. The fitness value for this run was the average score that the agent earned among all of the scenarios. After the first test, it was noted that letting the agent eat the power pills was diverting the behaviour from its objective, as some agents preferred to go to a different power pill or to eat another power pill during the edibility time in order to gain 50 points. For this reason, the second test began with the same scenarios, with one difference, namely that the game was set to begin while the ghosts were already edible and the given time was 300 time steps.

The output of this test was much better and demonstrated only one major disadvantage. In the real game, the agent faces a problem in which it eats a ghost and then follows another one, while the eaten ghost returns to the nest and regains its original predator status and starts chasing Pac-Man again. The problem was noticed when this ghost came between Pac-Man and the edible ghost; in this case, Pac-Man will commit suicide by charging directly at the ghost without noticing it, as it has not encountered enough problems like this in its training.

To solve this problem, a final set of 15 different scenarios was created. In some of these scenarios, the

agent begins at the position of a power pill and all of the ghosts are edible and in different positions in the maze. In two of the scenarios, the ghosts were very close to the agent and thus very easy to catch. In other scenarios, the agents was given the choice to go to the nearest ghost or to the next one, as the rest of the ghosts were very close to it. To solve the problem mentioned previously, several scenarios were created in which some of the ghosts are edible and the rest are not and are in positions that compromise the safety of the agent. These positions include cutting off the path between the agent and edible ghosts; thus, the only way for Pac-Man to score the possible 1400 points in that scenario was to go around the inedible ghost. Another situation was created in which chasing the closest ghosts (which is usually the obvious choice by GP) puts the agent into a very dangerous position.

Finally, the running time for this test was reduced to 200 time steps because in the full game, the edibility time decreases as the agent advances in the game. The rest of the GP parameters remained the same as in the escape test, with the exception of the fitness value, which here, is the average score achieved by the agent, rather than the average time steps.

VI. RESULTS AND DISCUSSION

Both the run of the standard GP and the final run of the training camp were tested under similar conditions with some exceptions including the action terminals in the function set, the population size and the number of generations. The changes in the population size and the number of generations were made to give both tests similar level of CPU time. However, predicting the required time to finish a GP run with Pac-Man simulators is rather difficult. The training camp GP runs have fairly consistent run-times due to the fixed number of time steps for each scenario. A single run of GP on the chasing scenarios takes on average 40 minutes while doing the same for the escaping scenarios takes around 5 hours. This difference in the time is caused by the type of terminals each run uses because the chasing agents seem to choose the terminals that will take the shortest path while the escaping agents choose the terminals that take the safest path and then require extra tests in order to find it.

The final run of the training camp (which combines the sub-behaviours) takes on average 1 hour to finish, so in total a single run of the training camp GP will take less than 7 hours to complete. With standard GP using the given parameters a single run requires approximately 12 hours to complete with various times range from 3 hours to 26 hours for a single run on the same machine and under the same conditions. This huge variability makes it impractical to exactly equalize the computational effort used by each approach.

The GP system was run 10 times in each case in order to obtain a clear view of what the agent is capable of achieving and in order to calculate various statistics. After the 10 runs were completed, the best individuals from each run were reused in the testing phase. Each of these individuals was given three lives and tested 100 times.

A. *The standard GP:*

The best (where best is defined as the individual with the best average) individual out of the 10 runs scored an average of 10,848 points over 100 tests and a maximum score of 18,280 points.

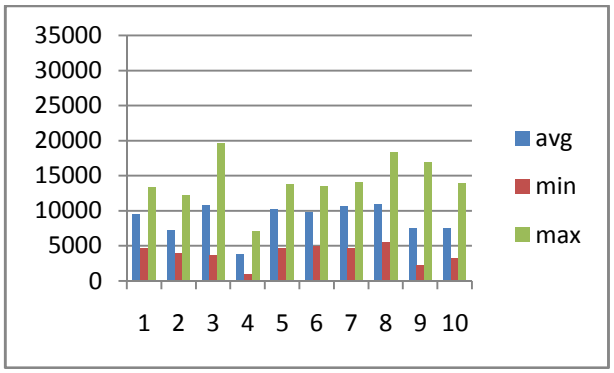

Figure 2: The results for the Standard GP 10 runs

This controller followed a strategy of eating pills when it is not in danger and escaping the ghosts when they come close. Of interest was its method of escaping the ghosts. It benefited from a terminal called *ToGhostSafe*, which takes Pac-Man to the nearest ghost by means of a safe route. This terminal succeeded in this mission because the direct route to the ghost is not safe because it leads directly to it, whereas turning around and proceeding by a route to come to the ghost from behind is considered safe. Thus, this method was successful at making the agent avoid the ghosts. In the expression tree, the first part indicates whether or not Pac-Man is in danger. If in danger, Pac-Man will use the *ToGhostSafe* terminal to escape; otherwise, it will follow the nearest edible ghost, if there is one, or the nearest pill.

Of the 10 runs, the best maximum score was achieved by a different controller (19,660 points), whereas the worst score amounted to 920 points. The average fitness out of the 10 runs was 8,744 points.

B. *The Training Camp:*

The *clearing the maze* training was abandoned after several tests proved that the result always remains the same, namely to use the *ToPill* terminal.

The *chase* training ran for 15 scenarios with a possible average score of 1,387 points. 5 runs of the GP were able to produce agents with an average fitness of 1,044 points and a best fitness of 1,084 points, which were achieved in 200 time steps in every scenario. Figure 3 shows the evolution of best individual fitness and the generation's average fitness through the 50 generations.

The *escape* agents were produced after training them on 30 different scenarios and allowing them 100 time steps in each. The same holds true for the chase agents. The best sub-agents were produced from 5 runs of the GP. The average fitness for these 5 runs was 78.17 time steps, while the best agent was able to survive for an average of 79.43 time steps over the 30 scenarios. Figure 4 shows the evolution of the generation fitness and best individual fitness over the generations.

Both the results of the chase and escape agents were gained from a testing phase, in which each agent ran each scenario 100 times in order to produce the required data.

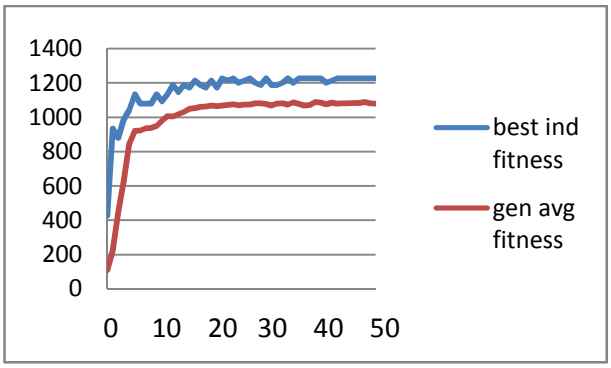

Figure 3: The evolution of the generation's average fitness and the best individual's fitness in the chase training camp

For comparison the 10 agents produced by standard GP were tested on the same scenarios used for the above training. In the chasing scenarios the agents scored an average of 524.6 while for the escaping scenarios the standard GP agents were able to survive for an average of 49.78 time steps per scenario. These results are significantly lower than the results obtained by the final controllers created by training Camp which scored an average of 812.8 points on the chasing scenarios and 71.79 time steps on the escaping scenarios.

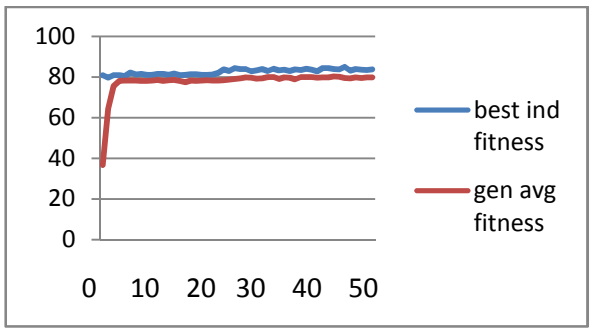

Figure 4: The evolution of the generation's average fitness and the best individual's fitness in the escape training camp

Finally, after finishing the sub-agent training, the top three produced by each technique in addition to the *ToPill* tree, which was produced in the clearing the maze training, were added as action terminals to the function set, and all of the other terminals in this category were eliminated. Eliminating the rest of the terminals came after several tests with all the action terminals, the most relevant terminals and the created sub-agents only. The latter group with only the new controllers as action terminals proved to be able to achieve the best results.

The explanation behind this may be the relatively small size of population which pushes GP to focus on the important functions only, reducing the amount of error and enhancing the final results. In this training camp run, the GP was forced to choose from 8 different terminals that had previously been carefully tested, and the margin of selecting a wrong terminal was quite low. In the first run with the full action terminals list, there were 24 terminals, meaning that choosing an incompatible terminal during the offspring creations was highly possible.

The GP system was run 10 times for the final run to create the full controller. The average fitness of the 10 runs was 10,574 points, around 1,800 points greater than the average of the standard GP while, the maximum fitness was 11,650 points, nearly 800 points better than the standard GP (see Table I).

TABLE I

The statistics for the 10 runs of the standard GP and the training camp

	avg	min	max	std err
Std GP	8,744	3,724	10,848	716
Training Camp	10,574	8,925	11,650	322

Analysing at these results from another perspective, Figures 2 and 5 reveal that 6 out of the 10 runs produced an average score of over 11,000 points in the training camp, while no individual was able to achieve this level in the standard GP. Also, 7 out of 10 runs produced a best score of over 20,000 points, including a best score of 31,850 points, with an average maximum score of over 22,000 points for the 10 runs.

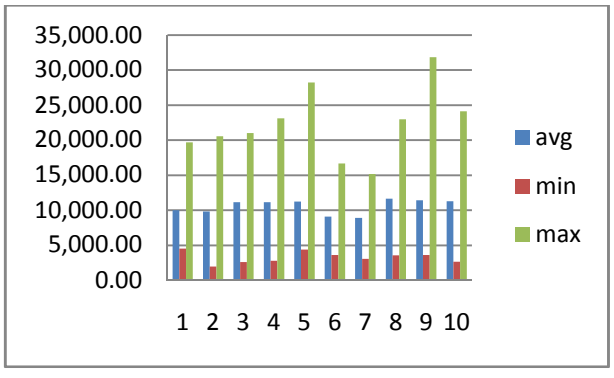

Figure 5: The results for the Training Camp 10 runs

The best controller from the training camp achieved 11,650 points, but it was not the controller with the best maximum score, which was 31,850 points. The agent that achieved this score had an average of 11,413 points. Most of the controllers that achieved the best

results followed the classic strategy of eating the pills while it is safe; if there was a danger, it would go to a power pill, if available, or select the escaping technique that it had acquired in training. Similarly, after eating a power pill, a chasing method was activated. Figure 6 shows the evolution of the best individual and the generation's average fitness over the 50 generation that created the most successful agents in this category.

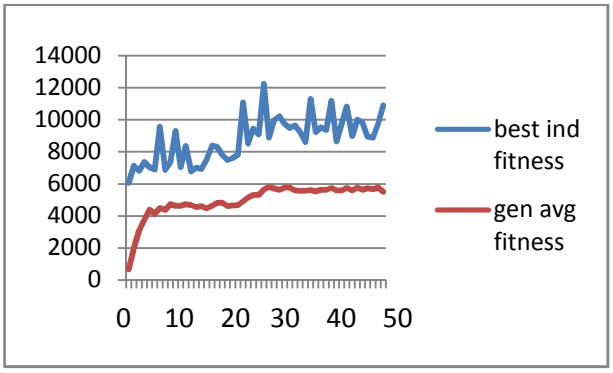

Figure 6: The evolution of the best individual's fitness and the generation's fitness

VII. CONCLUSION

From what is shown above, the training camp has proven to be capable of outperforming the standard GP. The training camp obtained an average score over 11,000 in 60% of the tests with a best score of 11,650, whereas standard GP was did not exceed 11,000 points and only exceeded 10,000 points 4 times, with a best average of 10,848 points. Moreover, in terms of the maximum achievement, the training camp reached a level of over 31,000 points, around 12,000 points more than standard GP, and in 7 times out of 10 the maximum score was over 20,000 points. The average of the highest scores achieved by the training camp was 22,340 points, while the standard GP had an average maximum score of 14,255 points with a best maximum of 19,660.

Using the training camp ensured the stability of the results. The worst average score achieved by the training camp was 8,925 points, just above the average fitness of the standard GP.

Finally using the average score of the 10 best individuals of both training camp and standard GP (table I); the *t-test* proved that these results are statistically significant with a *p* value of 0.0314.

VIII. FUTURE RESEARCH

The training camp idea offers an interesting area of research. A possible upgrade of the work done up to this point is to incorporate hierarchal learning. Due to the differences in the fitness functions, this would require applying a multi-objectives algorithm. Other important research is required to overcome the limitation of the scenarios and the effort involved in creating them by establishing an automated method by which to create and test the scenarios. One way to do this is to record all the game states during each game and after an interesting event occurs, backtrack to a previous suitable game state and select that as a scenario. Finally, the current system did not achieve the saving on CPU time that we had initially hoped for, due to each scenario being run for an average of a couple of hundred time steps. It would therefore be interesting to develop a set of single-action scenarios, where for each scenario, an agent is tested on the basis of the action it returns for a single game state.

VIII. REFERENCES

1. Jackson D, Gibbons A. Layered learning in boolean GP problems. Gen Program. 2007: 148-159.
2. Koza J. Genetic programming: on the programming of computers by means of natural selection. Cambridge, MA: The MIT Press; 1992.
3. Bajurnow A, Ciesielski V. Layered learning for evolving goal scoring behaviour in soccer players. Evol Comp. 2004.
4. Hsu WH, Gustafson SM. Genetic programming and multi-agent layered learning by reinforcements. Genetic and Evolutionary Computation Conference. Morgan Kaufmann; 2002.
5. Gustafson S, Hsu W. Layered learning in genetic programming for a cooperative robot soccer problem. Gen Program. 2001: 291-301.
6. Alhejali AM, Lucas SM. Evolving diverse Ms. Pac-Man playing agents using genetic programming. Workshop on Computational Intelligence (UKCI). 2010.
7. Lucas S. Evolving a neural network location evaluator to play ms. pac-man. Proceedings of the IEEE Symposium on Computational Intelligence and Games. Essex, UK: IEEE. 2005.
8. Emilio M, et al. Pac-mAnt: Optimization based on ant colonies applied to developing an agent for Ms. Pac-Man. IEEE Symposium on Computational Intelligence and Games (CIG). 2010.
9. Bell N, et al. Ghost direction detection and other innovations for Ms. Pac-Man. 2010 IEEE Symposium on Computational Intelligence and Games (CIG). 2010.
10. Keunhyun O, Sung-Bae C. A hybrid method of Dijkstra algorithm and evolutionary neural network for optimal Ms. Pac-Man agent. Second Word Congress on Nature and Biologically Inspired Computing (NaBIC). 2010.
11. Tong BKB, Sung CW. A Monte-Carlo approach for ghost avoidance in the Ms. Pac-Man game. International IEEE Consumer Electronics Society's Games Innovations Conference (ICE-GIC). 2010.
12. Robles D, Lucas S. A Simple Tree Search Method for Playing Ms. Pac-Man. IEEE Symposium on Computational Intelligence and Games (cig'09). 2009.
13. Samothrakis, S., D. Robles, and S.M. Lucas. *Fast Approximate Monte Carlo Tree Search Using Max-N Trees to play Ms Pac-Man* in IEEE Transactions on Computational Intelligence and AI in Games, to appear
14. Langdon W, Poli R. Evolutionary solo pong players. IEEE Congress on Evolutionary Computation. 2005.
15. Benbassat A, Sipper, M. Evolving Lose-Checkers players using genetic programming. IEEE Symposium on Computational Intelligence and Games (CIG). 2010.
16. Ebner M, Tiede T. Evolving driving controllers using genetic programming. IEEE Symposium on Computational Intelligence and Games. 2009.
17. Luke S. Genetic programming produced competitive soccer softbot teams for robocup97. Gen Program. 1998: 214–222.