Evil Evolves: Improving Generalized Flocking Strategies through Genetic Algorithm for the Ghost Team in the Game of Ms. Pac-Man

No Author Given

No Institute Given

Abstract. In this paper, we present an application of Genetic Algorithms and Flocking Strategies to control the Ghost Team in the game Ms. Pac-Man. Flocking Strategies allow to devise controllers having reduced computational complexity while generating emerging intelligent behavior. The Ghost Team controllers are optimized by means of Genetic Algorithms to be robust with respect to the stochastic elements of the game and the be effective against different possible opponents. The performance of the methodology proposed is tested and compared with that of other controllers.

Keywords. Flocking, Genetic Algorithms, Artificial Intelligence, Ms. Pac-Man, Video Games, Evolutionary Computation.

1 Introduction

Videogames such as Ms. Pac-Man are an ideal testbed for computational intelligence as they allow for the confrontation of multiple intelligent agents in a simple, yet challenging, context. The Ms. Pac-Man vs Ghosts competition has been run since 2009. Participants can submit controllers for either Ms. Pac-Man or for the Ghost Team.

In this work, an evolved controller for the Ghost Team based on Genetic Programming and Flocking Strategies is proposed. Flocking Strategies consist of simple rules for the interactions of the agents. Despite being computationally inexpensive, they often result in complex emerging intelligent behaviors. We coupled Flocking Strategies with Genetic Algorithms to design Ghost Team controllers that are effective at minimizing Ms. Pac-Man final score and that are also robust with respect to the stochastic elements of the game.

Although most of the research in this field focused on the design of controllers for Ms. Pac-Man, a number of works regarding the Ghost Team have been proposed. Nguyen and Ruck [1,2] present a controller based on Monte Carlo Tree Search where the behavior of Ms. Pac-Man is simulated. Their controller won the first Ms. Pac-Man Versus Ghost Team Competition held in 2011. Svensson and Johansson [3] exploit the behavior emerging capabilities of Influence Maps. A different line of research is pursued by Sombat *et al.* [4] that analyze Ms. Pac-Man matches to classify Ghost Team controllers according to their enjoyability and, therefore, understand the attribute that a NPCs should posses for players to be engaged.

The rest of the paper is organized as follows. The next section presents the Ms. Pac-Man problem description, including the competition features and restrictions. Some preliminary concepts and background of the work, along with related literature is commented in Section 3. Subsequently, Section 4 introduces the ghosts' AI approach which will be analyzed in the paper. Then the set of experiments conducted to perform this analysis is presented in Section 5. Finally, Section 6 describes the reached conclusions and outlines some future lines of work.

2 Ms Pac-Man. The Game and the Problem

The game of Pac-Man needs no presentation. Since its release in 1980 many variants have been proposed and Ms. Pac-Man was one of them. Released in 1981, Ms. Pac-Man presented several features that extended on the original game, such as a female character, new maze designs and several game-play changes. A screen-shot of the game is shown in Figure 1.

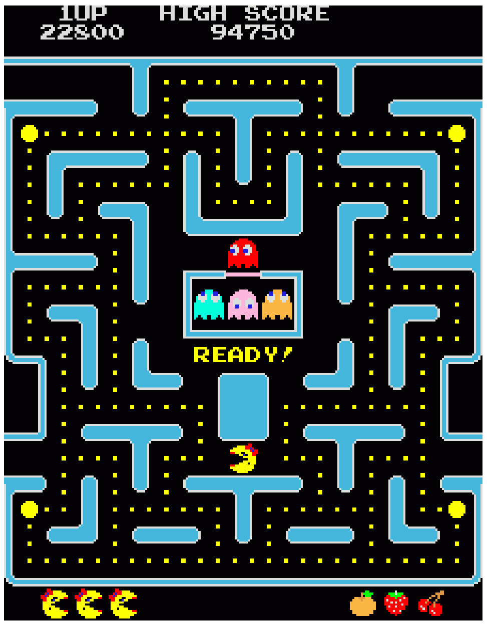


Fig. 1. A screenshot from the Ms Pac-man game.

In this game, Ms. Pac-Man has to collect all the pills in the maze while avoiding the four ghosts chasing her. If Ms. Pac-Man is reached by a ghost the player loses one life, Ms. Pac-Man is relocated at the initial position, and the ghosts respawn from the center of the maze. The powerpills located in the corners of the maze turn the ghosts edible for a short time, allowing Ms. Pac-Man to eat them. When a ghost gets eaten, it disappears from the game and respawns at the center of the maze after a certain amount of time. As the levels are cleared, the game becomes more difficult by changing certain parameters such as respawn time, length of time the ghosts are edible, and ghosts speed. Differently from the game of Pac-Man, this game has elements of randomness, firstly introduced to make the game more engaging.

Given its multiple challenges, the game has been chosen for the Ms. Pac-Man vs Ghosts competition[1], a game AI competition where participants can submit controllers for both Ms. Pac-Man and the Ghost Team. The aim of Ms. Pac-Man agents is to maximize the final score, while the aim of Ghost Team controllers is to minimize it. The version of the game implemented for the competitions differs slightly from the original one. A thorough description of the game rules can be found in [5]. For the purposes of this work, the relevant restrictions for the Ghost Team are briefly enlisted in the following:

– A ghost can choose its direction only at a junction. As a result, a ghost cannot turn back on itself.
– At every tick of the game all the ghosts reverse their direction according to a small random probability, set in the game implementation to 0.005.
– After 2000 game ticks, a level is considered completed: Ms. Pac-Man is rewarded the points of the remaining pills and the game moves on to the next level.

3 Preliminary Concepts and Background

In this section, the main techniques applied in the development of this work are briefly described, along with some relevant bibliography.

3.1 Swarm Intelligence and Flocking

Swarm intelligence (SI) is the term used to describe the type of coordinated intelligence that arises from the collective behavior of decentralized, self-organized systems, either natural or artificial [6]. SI techniques have been widely used in many fields including medicine, robotics, defense, astronomy, optimization, telecommunication, art, cinematography, and videogames.

Flocking refers to a SI technique proposed by Reynolds [7] for the coordinated movement of multiple AI agents. Originally, flocking algorithms have been developed to mimic lifelike behaviors of groups of beings such as herds of animals, flocks of birds, swarms of insects, and schools of fishes. A flocking system typically consists of a population of simple *agents* (or *boids*) interacting locally with one another depending on the distance between them. The agents follow very simple steering behaviors:

[1] http://www.pacman-vs-ghosts.net/, last visited on October 19, 2013

– *Separation* makes the agent steer away from close flock mates.
– *Alignment* makes the agent steer toward the average heading of the flock.
– *Cohesion* makes the agent steer toward the average position of distant flock mates.

Despite the lack of a centralized control structure dictating how individual agents should behave, the interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents [8]. Given this desirable property, the easiness of implementation, and the reduced computational cost, flocking algorithms have been extensively applied to videogames [9] and many other fields, such as cinematography, art, medicine, etcetera. Nevertheless, to the best of the authors knowledge this is the first work to actually applying flocking algorithms to the game of Pac-Man.

3.2 Genetic Algorithms

Evolutionary Algorithms (EAs) are a class of direct, probabilistic search and optimization algorithms gleaned from the model of Darwinistic evolution [10]. They have been widely used for solving combinatorial and optimization problems.

The most extended class of EA are the Genetic Algorithms (GAs) [11]. A GA is composed of a *population of candidate solutions* to a problem that evolve towards optimal (local or global) points of the search space by recombining parts of the solutions to generate a new population. The decision variables of the problem are encoded in strings with a certain length and cardinality. In GAs' terminology, these strings are referred to as *chromosomes*, each string position is a *gene* and its values are the *alleles*. The alleles may be binary, integer, real-valued, etc, depending on the codification (which in turn may depend on the type of problem). As stated, the "best" parts of the chromosomes (or building-blocks) are guaranteed to spread across the population by a *selection mechanism* that favors better (or fitter) solutions. The quality of the solutions is evaluated by computing the *fitness* values of the chromosomes; this fitness function is usually the only information given to the GA about the problem.

EAs have been applied in a wide range of optimization problems, including the videogames area [12]. The problem is they usually involve considerable computational cost and thus they are not frequently used in on-line games. In fact, the most successful proposals for using EAs in games correspond to off-line applications [13], that is, the EA works (for instance, to improve the operational rules that guide the agents' actions) while the game is not being played, and the results or improvements can be used later during the game. Through offline evolutionary learning, the quality of agents' intelligence in commercial games can be improved, and this has been proven to be more effective than opponent-based scripts.

In recent years, a number of EAs has also been proposed to address different aspects of the game of Pac-Man [14,15,16,17,18]. Thawonmas [19] applies a GA to optimize the parameters of the Pac-Man controller ICE Pambush 3, winner of the IEEE CEC 2009 Ms. Pac-Man competition. A number of authors made use of

EAs to evolve neural network-based controllers, both for Ms. Pac-Man [20,21,22] and the ghosts [23]. Gagne and Congdon [24] evolve rule-based intelligent agent for the ghost team. Finally, Cardona *et al.* [25] explore competitive co-evolution techniques to evolve at the same time Ms. Pac-Man and Ghost Team controllers.

In this work, an offline GA is applied to a flocking model for the Ghost Team, in order to improve its decision engine, which will be used later during game (online).

4 Ghost Team AI: Evolutionary Flocking

In this section the evolutionary flocking strategy model developed for designing controllers for the Ghost Team is described.

4.1 Generalized Flocking Strategies

We define a Flocking Rule, ϕ as a set of two vectors, ϕ^d and ϕ^m, that jointly describe the steering behavior of a ghost under certain conditions. Each FR considers a number N of concentric neighborhoods centered on the ghost. The limits of each neighborhood are specified by vector $\phi^d \in \mathbb{N}^N$; please note that the elements of vector ϕ^d are always sorted in ascending order as they represent the radii of the concentric neighborhoods . The last element of this vector, ϕ^d_N, is set to ∞ by default to avoid feasibility problems. Vector $\phi^m \in [-1, 1]^N$ defines the magnitude of the steering force applied on the ghost when an agent falls into one of the neighborhoods. As an example, let us consider a scenario with two ghosts, Ghost A and B, and the following flocking rule for Ghost A:

$$\phi^d = \{10, 20, \infty\}$$
$$\phi^m = \{-0.5, 0, 0.3\}$$

This sample rule considers only three neighborhoods ($N = 3$). Ghost A is located at position $v_a = (1, 0)$ while Ghost B is located at position $v_b = (5, 3)$. Let us find the steering force on Ghost A resulting from the interaction with Ghost B. First, the difference vector v_δ and the Euclidean distance δ between the two ghosts are calculated:

$$v_\delta = v_b - v_a \tag{1}$$
$$\delta = \|v_\delta\| \tag{2}$$

In this example, $v_\delta = (4, 3)$ and $\delta = 5$. To identify the magnitude ϕ^m_N we need to determine the neighborhood n where Ghost B belongs, by applying the following condition:

$$\phi^d_{n-1} < \delta \leq \phi^d_n \tag{3}$$

where $\phi^d_0 = 0$. In this case, δ is less than 10, therefore Ghost B falls into the first neighborhood ($n = 1$) and the magnitude ϕ^m_n of the steering behavior is the first element of vector ϕ^m, $\phi^m_1 = -0.5$. As the magnitude is negative, the ghosts repel each other. If $delta \in (10, 20]$ the magnitude would have been 0 (no interaction), while if $\delta > 20$ the magnitude would have been 0.3 (attraction). Finally, the steering force on Ghost A resulting from the interaction with Ghost B is given by:

$$f^B = \phi^m_n \cdot \frac{v_\delta}{\delta} \tag{4}$$

In the example, $f^B = (-0.4, -0.3)$.

From the example it can be easily seen that a negative magnitude corresponds to the behavior of separation, while a positive magnitude corresponds to the behavior of cohesion. No alignment behavior is included in this strategy model for the Ghost Team as it would make the controllers very predictable.

Differently from the basic flocking algorithm where only one type of agent is considered, in the game Ms. Pac-Man a variety of different actors are present. Also, the ghosts can be in different states (e.g., normal or edible). To be effective, a strategy for the Ghost Team should take into account at least all the elements presented to the player on the screen. Let S be the set of possible ghost states:

$$S = \{\text{HUNTER}, \text{HUNTED}, \text{BLINKING}\}$$

$HUNTER$ is the "normal" state of a ghost (i.e., kills Ms. Pac-Man on contact). When Ms. Pac-Man eats a powerpill all the ghosts become $HUNTED$ for a certain length of time (i.e., is killed by Ms. Pac-Man on contact). When this period is about to expire, the ghost blinks to warn the player; we call this state $BLINKING$. Let A be the set of all the actors in the game:

$$A = \{\text{PACMAN}, \text{POWERPILL}, \text{HUNTER}, \text{HUNTED}, \text{BLINKING}\}$$

$HUNTER$, $HUNTED$, and $BLINKING$ refers to ghosts in that state. We can now define a Flocking Strategy (FS) for the Ghost Team, $\mathit{\Phi}$, as:

$$\mathit{\Phi} : S \times A \rightarrow \phi$$

A Flocking Strategy is a function that, given a ghost state and the type of actor considered, returns the flocking rule that has to be applied to calculate the steering force on the ghost resulting from the interaction with the actor.

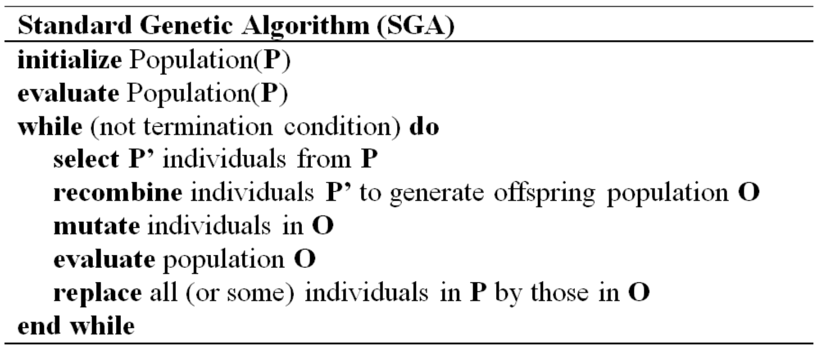
Every time a ghost is at a junction the game needs to calculate its next move. The controller based on the flocking strategy provides the next move by following these steps:

1. The status s of the current ghost G is determined.
2. For all the actors a in the game (i.e., Ms. Pac-Man, remaining powerpills, ghosts excluding the current one) execute the following steps:
   (a) Determine the Flocking Rule $\phi = \mathit{\Phi}(s, a)$ to be applied.
   (b) Calculate the difference vector v_δ (1) and the Euclidean distance δ (2) between the current ghost and the actor.

   (c) Find the neighborhood n and the magnitude ϕ_n^m according to Flocking Rule ϕ (3).
   (d) Finally, compute the steering force f_a applied on the ghost resulting from the interaction with actor a (4).
3. Calculate the total steering force as the sum of all the steering forces: $f = \sum_a f^a$.
4. The steering force can be easily translated in a ranking for the next ghost direction:
   – UP $= -f_2$.
   – DOWN $= f_2$.
   – LEFT $= -f_1$.
   – RIGHT $= f_1$.
5. Choose the possible direction (see restrictions in Section 2) having maximum rank.

4.2 Improving Flocking Strategies by Means of GAs

In this work we are dealing with a two-player competitive game with stochastic elements. A complete definition of the game's extensive form would allow us to define a set of non-dominated strategies. Sadly, given the intrinsic complexity of the game, this approach is prohibitive. On the other hand, a Flocking Strategy could be manually designed by an expert with decent results. Nevertheless, given the high number of parameters, it is desirable to automatize the definition of an effective strategy by means of an optimization algorithm. Given the lack of the game extensive form and the reduced number of constraints involved, GAs appear to be a sensible choice. A standard GA's procedure is shown in Figure 2.



```
Standard Genetic Algorithm (SGA)
initialize Population(P)
evaluate Population(P)
while (not termination condition) do
    select P' individuals from P
    recombine individuals P' to generate offspring population O
    mutate individuals in O
    evaluate population O
    replace all (or some) individuals in P by those in O
end while
```

Fig. 2. Standard GA pseudocode.

In the following, the elements comprising the GA implemented to optimize flocking strategies for the Ghost Team are described.

Initial Population To provide the highest degree of genetic variability to the GA, the initial population is created as a random set of flocking strategies defined as:

$$\forall s \in S, a \in A, i = 1, \ldots, N, \quad \phi_i^d \sim U(\phi_{i-1}^d, \infty) \tag{5}$$

$$\forall s \in S, a \in A, i = 1, \ldots, N, \quad \phi_i^m \sim N(0, 1/3), \phi_i^m \in [-1, 1] \tag{6}$$

The elements of ϕ^d have a uniform distribution, while the elements of ϕ^m have a truncated normal distribution. The parameters of the normal distribution have been set so as to generate most of the magnitudes close to zero.

Fitness Function The definition of the fitness function is one of the most critical aspects in a GA. The optimization algorithm proposed should generate Ghost Teams strategies that perform well against any possible Ms. Pac-Man strategy and, at the same time, are resilient to the random ghosts direction reverse events (see Section 2). To achieve this result, each flocking strategy is pitted against two different Ms. Pac-Man controllers, included in the Ms. Pac-Man vs Ghosts competition framework: *StarterPacMan* (SPM) and *NearestPillPacMan* (NPPM). The game is simulated 30 times for each Ms. Pac-Man controller. Thanks to that we can take advantage of the central limit theorem to compute a relatively precise 95% confidence interval of the final score obtained by the Ms. Pac-Man controllers. Let μ_{SPM}, σ_{SPM}, μ_{NPPM}, σ_{NPPM} be the average score obtained by controller SPM in the 30 runs, the standard deviation of the SPM's scores, the average score obtained by controller NPPM in the 30 runs, and the standard deviation of these scores, respectively. The upper limits of the confidence intervals for the scores of the two controllers are:

$$\overline{\text{CI}}_{\text{SPM}} = \mu_{\text{SPM}} + Z \cdot \frac{\sigma_{\text{SPM}}}{\sqrt{30}}$$

$$\overline{\text{CI}}_{\text{NPPM}} = \mu_{\text{NPPM}} + Z \cdot \frac{\sigma_{\text{NPPM}}}{\sqrt{30}}$$

where Z is the 95% percentile of the standard normal distribution (i.e., 1.959963985 approx). The upper limits of the confidence intervals represent the maximum score that the Ms. Pac-Man controllers should get 95% of the times. Our objective is to obtain a Ghost Team controller that minimizes these two values. The fitness function can be now defined as:

$$\text{FITNESS} = \frac{0.7}{\overline{\text{CI}}_{\text{SPM}}} + \frac{0.3}{\overline{\text{CI}}_{\text{NPPM}}}$$

The fitness function is basically the weighted average of the confidence intervals' inverses. A bigger weight has been assigned to SPM as it is more complex than NPPM in terms of behavior.

Selection, Recombination, and Mutation After all the strategies of the current generation have been evaluated, it is possible to generate the individuals of the next generation. For each strategy Φ to be generated, two flocking strategies Φ_1 and Φ_2 are chosen by *roulette-wheel selection* (i.e., every individual has a probability of being chosen proportional to its fitness). Strategy Φ is created by random recombination of the parameters of Φ_1 and Φ_2:

$$\forall s \in S, a \in A, i = 1, \dots, N, \quad \phi_i^d = \text{rand}(\phi_i^d \in \Phi_1(s,a), \phi_i^d \in \Phi_2(s,a))$$

$$\forall s \in S, a \in A, i = 1, \dots, N, \quad \phi_i^m = \text{rand}(\phi_i^d \in \Phi_1(s,a), \phi_i^d \in \Phi_2(s,a))$$

where *rand* is a function that returns a random value chosen among its arguments. During the recombination, mutations can occur with probability p^{mut}. When a mutation happens, the current element is re-initialized to a random value, according to Formulas (5) and (6).

The mutation probability p^{mut} is determined dynamically. Initially, its value is set to $p^{\text{mut}} = 0.00125$. At each iteration t, its value changes depending on the coefficient of variation of the current population fitness, c_v:

$$p_t^{\text{mut}} = \begin{cases} 0.00125 & \text{if } c_v > 0.6 \\ p_{t-1}^{\text{mut}} & \text{if } 0.3 < c_v \leq 0.6 \\ 2 \cdot p_{t-1}^{\text{mut}} & \text{if } 0.2 < c_v \leq 0.3 \\ 4 \cdot p_{t-1}^{\text{mut}} & \text{if } 0.1 < c_v \leq 0.2 \\ 8 \cdot p_{t-1}^{\text{mut}} & \text{if } c_v \leq 0.1 \end{cases}$$

c_v measures the degree of variability of the population in terms of fitness. When the variability is low, we increment the mutation probability to introduce new chromosomes in the genetic pool of the population. When the variability is too big, the mutation probability is set to a low initial value.

Once the recombination is done and the new Flocking Strategy is created, all its vectors ϕ_i^d and ϕ_i^m are sorted in ascending order with respect to the values of ϕ_i^m, to preserve the feasibility of the Flocking Rules.

5 Experiments and Results

In the experiments, the GA described in the previous chapter has been run for 50 generations with a population of 50 candidate strategies. At each iteration, the next generation was constituted by 49 recombined individuals plus the best solution of the current generation.

All the algorithm have been implemented in Java within the framework provided for the Ms Pac-Man vs Ghosts competition. The final program run on a Intel(R) Core(TM) i5-2500K @ 3.3GHz with four cores and 4GB of RAM. Each experiment run on a single core and made use of less than 300MB of memory.

The first experiment regards the comparison of the performance of the Ghost Team controllers obtained with different values of the parameter N (i.e., the number of neighborhoods considered in the Flocking Rules). The best strategy found with a certain number of neighborhoods should be at least as good as those found with a lower number of neighborhoods, as the solution space of the former is bigger and contains the others'. Nevertheless, as the number of neighborhoods increases, the solution space increases substantially, that in turn could affect the actual performance of the GA. In the experiments we use the inverse fitness, FITNESS^{-}1, as a measure of performance as it is easier to interpret than the fitness function that takes values close to zero. Being the inverse of the fitness value, a lower FITNESS^{-}1 value corresponds to a better controller, and vice-versa.

	$N = 1$	$N = 2$	$N = 3$	$N = 4$	$N = 5$
Best FITNESS^{-}1	1,751.41	1,452.54	1,484	1,489.13	1,403.33
Avg. FITNESS^{-}1	1,870.75	1,633.08	1,586.84	1,545.49	1,689.20
Worst FITNESS^{-}1	1,980.36	1,737.86	1,712.54	1,633.03	1,959.05
Mean optimization time (s)	1,114	1,225	1,293	1,246	1,202

Table 1. Performance of the controllers for the Ghost Team obtained by the GA.

Table 1 shows the performance of the evolved controllers over three runs of the GA with $N = 1, \ldots, 5$. Each column is associated to a different number of neighborhoods. The first row displays the inverse fitness of the best individual found. The second rows presents the average controllers fitness, while the third illustrates the fitness value of the worst controller found. Finally, the last row reports the average optimization CPU time over the three runs. By observing the table the following conclusions can be drawn:

– The best controller is obtained with $N = 5$.
– By observing the Avg. FITNESS^{-}1 row it easy to see that, as the value of N increases, the average performance of the controllers tends to improve as well. The experiments with $N = 5$ represent an exception and this could be due to the increased complexity in the search space resulting from the high number of parameters to be optimized by the GA.
– For the fitness function considered, two neighborhood areas are sufficient to generate a very efficient controller.
– The gap between the best and the worst fitness in each column suggests that the problem presents many local optima in the solution space.
– Concerning the optimization time, it seems that it is basically unaffected by changes in the value of N. The small difference in the computational time might depend on factors such as memory and CPU usage by other processes.

When playing against the best Ghost Team controller obtained by the GA it is possible to observe complex behaviors. Figure 3 illustrates an example:

despite the lack of coordination between them, the ghosts managed to corral and surround Ms. Pac-Man that is now entrapped in a corridor.

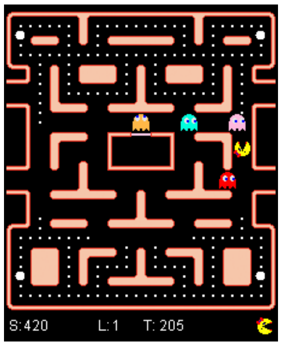


Fig. 3. Example of emerging behavior. The ghosts surrounded and entrapped Ms. Pac-Man.

In the next experiment, we compare our controllers with the five Ghost Team controllers included in the competition framework. Their FITNESS^{-}1 values, computed exactly as per the GA solutions, are the following:

- *AggressiveGhosts* : 4,645.79;
- *Legacy* : 4,577.61;
- *Legacy2TheReckoning* : 3,631.94;
- *RandomGhosts* : 12,205.60;
- *StarterGhosts* : 3,785.49;

According to these results, the best controller is *Legacy2TheReckoning*, followed by *StarterGhosts*. Nevertheless, their FITNESS^{-}1 value is more than three times bigger than that of the best evolved flocking strategy. These results support the claim that flocking strategies are a viable option for the definition of intelligent controllers.

6 Conclusions and Future Work

In this paper, a new controller for the Ghost Team based on flocking strategies is proposed. A GA is presented to design optimized strategies offline. The methodology has been empirically tested. The results show that flocking strategies, despite the reduced computational cost, model complex behaviors and produce effective and challenging agents. This work is just scratching the surface and there is still a lot to be investigated. Some possible future lines of research are highlighted in the following.

The fitness function can be easily extended by including more Ms. Pac-Man controller. This should result in a Ghost Team controller that performs better against a wider range of opponents. By considering in the GA fitness function the best Ms. Pac-Man controllers that took part to the competitions, it would be possible to generate Ghost Team controllers that are capable of tackling the best known Ms. Pac-Man strategies.

Moreover, it would be interesting to compare the controllers obtained by applying the presented methodology with the best Ghost Team controllers that took part to the Ms. Pac-Man vs Ghosts competition. This would allow us to really understand the limits of Flocking Strategies.

The GAs as a means to optimize Flocking Strategies have proven to be satisfactory. Nevertheless, the recombination step causes abrupt changes in the solutions' parameters and might generate individuals that are very different from the initial ones. It would be interesting to investigate the effectiveness of optimization methods that allows for small changes in the solutions parameters. Particle Swarm Optimization (PSO) algorithms might be a sensible choice. In fact, on top of making few or no assumptions about the problem, PSO algorithm are particularly effective with problems that are noisy and present many multiple optima, such as this one.

We hope that this work will be a useful source of ideas for future research on bio-inspired algorithms in games and will contribute further in the development and solution of more complex problems.

References

1. Nguyen, K., Thawonmas, R.: Applying Monte-Carlo Tree Search to collaboratively controlling of a Ghost Team in Ms Pac-Man. In: Proceedings of the 2011 IEEE International Games Innovation Conference (IGIC 2011). (2011) 8–11
2. Nguyen, K., Thawonmas, R.: Monte Carlo Tree Search for collaboration control of ghosts in Ms. Pac-Man. IEEE Transactions on Computational Intelligence and AI in Games **5(1)** (2013) 57–68
3. Svensson, J., Johansson, S.: Influence Map-based controllers for Ms. PacMan and the ghosts. In: Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG 2012). (2012) 257–264
4. Sombat, W., Rohlfshagen, P., Lucas, S.: Evaluating the enjoyability of the ghosts in Ms Pac-Man. In: Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG 2012). (2012) 379–387
5. Rohlfshagen, P., Lucas, S.: Ms Pac-Man versus Ghost Team CEC 2011 competition. In: Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC 2011), IEEE Press (2011) 70–77
6. Beni, G., Wang, J.: Swarm intelligence in cellular robotic systems. In: Robots and Biological Systems: Towards a New Bionics? Volume 102 of NATO ASI Series F: Computer and Systems Sciences. SPRINGER-Verlag (1993) 703–712
7. Reynolds, C.: Flocks, herds and schools: a distributed behavioral model. Computer Graphics **21(4)** (1987) 25–34
8. Spector, L., Klein, J., Perry, C., Feinstein, M.: Emergence of collective behavior in evolving populations of flying agents. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003). (2003) 61–73

9. Scutt, T.: Simple swarms as an alternative to flocking. In: AI Game Programming Wisdom. Charles River Media (2002) 202–208
10. Bäck, T.: Evolutionary algorithms in theory and practice. Oxford University Press (1996)
11. Goldberg, D., Korb, B., Deb, K.: Messy genetic algorithms: motivation, analysis, and first results. Complex Systems **3(5)** (1989) 493–530
12. Mora, A., Moreno, M., Merelo, J., Castillo, P., García-Arenas, M., Laredo, J.: Evolving the cooperative behaviour in Unreal^{TM} bots. In: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG 2010), IEEE Press (2010) 241–248
13. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Improving opponent intelligence through offline evolutionary learning. International Journal of Intelligent Games and Simulation **2(1)** (2003) 20–27
14. Gallagher, M.: Learning to play Pac-Man: an evolutionary, rule-based approach. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003). (2003) 2462–2469
15. Galván-López, E.: Comparing the performance of the evolvable πGrammatical Evolution genotype-phenotype map to Grammatical Evolution in the dynamic Ms. Pac-Man environment. In: Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010). (2010) 1–8
16. Alhejali, A., Lucas, S.: Evolving diverse Ms. Pac-Man playing agents using genetic programming. In: Proceedings of the 2010 UK Workshop on Computational Intelligence (UKCI 2010). (2010) 1–6
17. Brandstetter, M., Ahmadi, S.: Reactive control of Ms. Pac Man using information retrieval based on Genetic Programming. In: Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG 2012). (2012) 250–256
18. Alhejali, A., Lucas, S.: Using a training camp with Genetic Programming to evolve Ms Pac-Man agents. In: Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG 2011). (2011) 118–125
19. Thawonmas, R.: Evolution strategy for optimizing parameters in Ms Pac-Man controller ICE Pambush 3. In: Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games (CIG 2010). (2010) 235–240
20. Lucas, S.: Evolving a neural network location evaluator to play Ms. Pac-Man. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG 2005). (2005)
21. Burrow, P., Lucas, S.: Evolution versus Temporal Difference Learning for learning to play Ms. Pac-Man. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG 2009). (2009) 53–60
22. Keunhyun, O., Sung-Bae, C.: A hybrid method of Dijkstra algorithm and evolutionary neural network for optimal Ms. Pac-Man agent. In: Proceedings of the 2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC 2010=. (2010) 239–243
23. Jia-Yue, D., Yan, L., Jun-Fen, C., Feng, Z.: Evolutionary neural network for ghost in Ms. Pac-Man. In: Proceedings of the 2011 International Conference on Machine Learning and Cybernetics (ICMLC 2011). Volume 2. (2011) 732–736
24. Gagne, D., Congdon, C.: FRIGHT: A flexible rule-based intelligent ghost team for Ms. Pac-Man. In: Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG 2012). (2012) 273–280
25. Cardona, A., Togelius, J., Nelson, M.: Competitive coevolution in Ms. Pac-Man. In: Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC 2013). (2013) 1403–1410