Combining Cooperative and Adversarial Coevolution in the Context of Pac-Man

Alexander Dockhorn and Rudolf Kruse
Institute of Intelligent Cooperating Systems
Department for Computer Science, Otto von Guericke University Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany
Email: {alexander.dockhorn, rudolf.kruse}@ovgu.de

Abstract—**In this paper we discuss our recent approach for evolving a diverse set of agents for both the Pac-Man and the Ghost Team track of the current Ms. Pac-Man vs. Ghost Team competition. We used genetic programming for generating various agents, which were distributed in multiple populations. The optimization includes cooperative and adversarial subtasks, such that Pac-Man is constantly competing against the Ghost Team, whereas the Ghost Team is formed of four cooperatively evolving populations. For the generation of a Ghost Team and calculation of the associated fitness we took one individual from each population. This strict separation preserves the evolution pressure for each population such that respective Ghost Teams compete against each other in developing an efficient cooperation in catching Pac-Man. This approach not only is useful for developing a versatile set of playing agents, but also for adapting the team to the current behavior of the competing populations. Ultimately, we aim for optimizing both tasks in parallel.**

I. Introduction

Artificial intelligence (AI) in games proved to be successful in creating playing agents in a variety of games. Besides the well known successes in 2-player full-information games such as Chess [1] and Go [2], artificial agents were recently successful in playing games of Poker [3] against expert level players. All those agents feature top-level play against human players. However, the entertainment industry is also in need of intermediate solutions, which scale well with the level of human play they currently face. Thus, bringing both an enjoyable playing experience and providing a suitable opponent for mastering the game is a key demand for applications in the entertainment industry.

Nevertheless, designing multiple AIs is a cumbersome and expensive task for game developers. Simple solutions like conditionalized behavior, or cheating AIs are the result. Such an agent can be reported as unfair by the players and in many cases a winning strategy can be found to exploit the deterministic behavior of the implemented AI. Alternative to those classical methods, AIs based on Monte Carlo Tree Search are non-deterministic and were successfully applied in many games, e.g. Checkers [4], Backgammon [5], and Go [6]. However, scaling the skill level of such a heuristic search algorithm is unpredictable, which is rendering it less useful in playing with unexperienced players. For this purpose, we want to present our development process for creating diverse cooperating agents, which are able to adapt to the players current skill level.

This paper focuses on automatically learning playing agents for the Ms. Pac-Man vs. Ghost Team competition [7], the successor of the CEC 2011 competition [8]. Ms. Pac-Man is the unofficial second installment of the Pac-Man Series, which was originally released by Namco in the year 1980. In 1982 the studio General Computer Corporation made use of the successful game design and developed their own expansion of the game. Besides improvements in the level and graphic design the game featured a variable Ghost behavior. While the original game included deterministic Ghosts with differing functions, a degree of randomness was added in Ms. Pac-Man. This avoided exploitation by players, who knew the perfect path for collecting all the pills without dying, and therefore made the game much more interesting in the long term.

Our learning process is based on a combination of cooperative and adversarial coevolution with multiple populations. We expect the strict separation to support diversity in the overall team. In the original Pac-Man game the four Ghosts were either set to strictly follow the player, cut off his escape route or guarding a specific area. Those strategies are easily translated to conditionalized behaviors, which will form the basis of our gene pool. We will use genetic programming to create and modify agents and rate them based on their cooperative success in catching Ms. Pac-Man. Besides evolving a Ghost Team, we are also learning an agent for Pac-Man. This should simulate the learning process of an actual player. Both learning processes are individually evaluated based on their performance against hand-coded bots. Furthermore, their populations are combined in an adversarial learning scheme implementing coevolution.

The remainder of this paper will be structured as follows: in Section II & III we will present an overview about the game, the competition task and review past submissions. Furthermore, we will shortly review genetic programming and its recent applications. After this short introduction, we will present our approach for developing a diverse set of playing agents for Pac-Man using genetic programming in Section IV. The concept of our combined coevolution will be introduced in Section V. In the following evaluation section (Section VI) we will first confirm successful adaptation in both single learning subtasks. We will go on with evaluating the behavior of the adversarial coevolution of both agents. This paper ends with a discussion of our results in Section VII after which we will provide a conclusion regarding the general applicability of our approach.

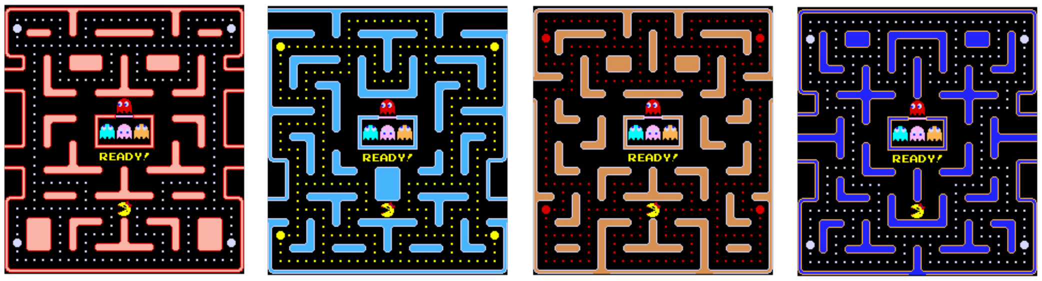


Fig. 1: The four mazes included in the Ms. Pac-Man vs. Ghost Team competition

II. The Game

The current installation of the Ms. Pac-Man versus Ghost Team Competition started in 2016 [7]. A simulator of the game is available through the competition website [9]. The competition is divided into two tracks, one for implementing a Pac-Man AI and the other one for implementing a full set of Ghost controllers.

The aim of Pac-Man is to traverse a maze and collect all the pills, while avoiding contact with the four Ghosts. Special power pills are distributed in the corners of each level. Collecting them allows Pac-Man to slow down his enemies and eat them. Each of these actions scores Pac-Man points. The scoring scheme is as follows:

- **Eating a pill:** 10 points. Each of the 4 mazes includes about 200 pills.
- **Eating a power pill**: 50 points. Each maze contains 4 power pills in the corners.
- **Eating a Ghost:** After eating a power pill, Ghosts will be eatable for a short period. Eating multiple Ghosts per period, will score Pac-Man 200, 400, 800, and 1600 points per Ghost. However, eating another power pill will reset the combo counter. The high risk in pursuing Ghost in contrast to the high reward makes eating Ghosts an interesting challenge of the game.

A level ends after collecting all the pills and power pills. The maximal score per level is equal to:

$$score_{max} = 10n + 4 \times 50 + 4 \times (200 + 400 + 800 + 1600)$$

where n equals the number of pills per level. Completing a level will reset all positions and load the next level cycling through all four levels. The game goes on till Pac-Man lost all of his 3 lives. Agents can be rated by the highest or the average number of points they score.

The Pac-Man controller needs to respond to all $getMove()$ queries with a timebudget of 40 *ms*. The same timebudget applies to the Ghost Team controller, which can freely distribute the time between all four Ghosts. Since Ghosts can only change their direction at junctions of a maze, the free distribution of time ensures that a single Ghost controller will have enough time to respond to the current game state at junctions.

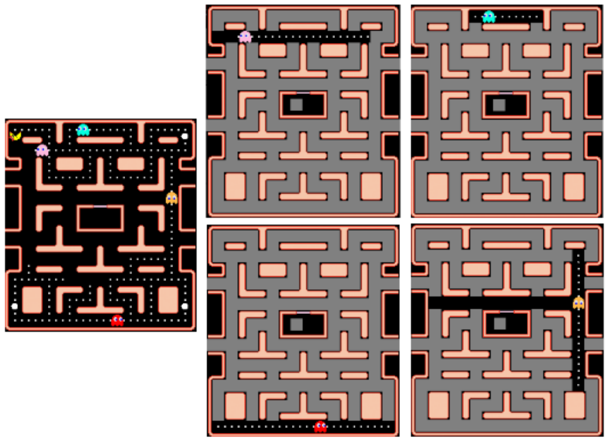


Fig. 2: A normal game view and the partial observation scheme of the Ghosts.

A. Partial Observation

In contrast to the original game, the competition impairs agents in their observation of the world. All query responses to the game's application programming interface (API) are limited to the current visibility of objects from the agents position. Here, walls are blocking the perception, therefore, limiting the agent to notice objects only if they are in an orthogonal line of sight. See fig. 2 for a comparison of the standard game view and the implemented line of sight for partial observability.

While general information about the maze is permanently available, the availability of pills and power pills, and the positions of moving actors need to be tracked throughout the game. Furthermore, Ghosts are limited in their communication, such that they are only able to send messages including their own position, their target position, or the recently observed position of Pac-Man.

III. Previous Work

Since the competition is in its second installment, many researchers already crafted solutions for learning agents who are able to play the game on a fairly good level. We want to give a short overview of used strategies for implementing either a Pac-Man or a Ghost Team AI. Since the design of the competition changed throughout the years, results are not always comparable. For example the competition in 2011 was based on pixel map input, which is much more abstract than the current year's API. However, partial observation adds another interesting challenge to the game and, for example, makes it hard to plan several steps ahead.

Since the dawn of applying computational intelligence in games, using rule based agents is a common approach applicable to most games. Here, expert knowledge can be integrated to develop agents with high skill level. Inductive learning can be used to develop such rules from random playouts and, therefore, eliminate the need of human guidance. Gallagher and Ryan [10] used population based incremental learning to adapt a finite-state machine for playing a simplified version of Pac-Man. In each state the appropriate move was chosen based on an associated probability table. The proposed solution was only a minor success due to the lack of an efficient state representation. Extending the solution to the full game was judged to be impractical.

In contrast, simulation based methods like Monte-Carlo Tree Search (MCTS) proved to be useful in previous years of this competition. The MCTS approach by Tong and Sung [11] was capable of avoiding Ghosts and attaining a maximal score of 21.000 points. Robles and Lucas [12] applied a Tree Search method at the screen capture version of the game. Together with Samothrakis they implemented Ghost Team agents using MCTS [13]. The same approach was used by Nguyen and Thawonmas [14] to create a full Ghost Team. In their simulations the Ghosts' search tree was expanded randomly while Pac-Man moved according to simple rules. This approach outperformed all other candidates and won the CEC 2011 competion [8]. Ikehata and Ito [15] used MCTS for creating a Pac-Man AI named ICEPambush3, which outperformed the CIG 2009 winner.

In contrast to simulation based methods, Lucas [16] used Neural Networks to evaluate the current game state and report an appropriate move. Gallagher and Ledwich [17] adopted a similar method using the visual output of the game. Due to the high complexity of the neural network input they determined the network weights using a neuroevolutionary approach. Although the average score was lower, it proved to be capable of learning skillful playing behavior in context of complex inputs.

Genetic programming is a third concept often applied for creating agents for the game Pac-Man. Here, each individuum encodes a conditionalized tree, which when evaluated returns an appropriate action. Mutation and crossover operators can be defined to evolve the tree and therefore change the behavior of the developed agent. We chose our tree-representation based on the work of Kruse et al. [18]. John Koza [19], Alhejali and Lucas [20], as well as Brandstetter and Ahmadi [21] proved the capabilities of genetic programming in the context of Pac-Man.

In addition to genetic programming, learning a full set of Ghosts can be done using coevolution. Here the genetic representation of the Ghost Team is split into several parts, which need to cooperate in order to catch Pac-Man. A general overview of coevolution algorithms was presented by Wiegand [22]. In this paper we will further expand on the work of Cardona et al. [23], who made use of the coevolution framework to learn a set of Ghost controllers.

IV. Generating Behavior Trees through Genetic Programming

As it was already suggested in John Koza's book about genetic programming [19] developing an agent for Pac-Man can be done by evolving conditionalized trees. Furthermore, our work was inspired by Alhejali and Lucas [20] who evolved Ms. Pac-Man agents in the previous competition. The full source code of our AI will be made available on our website [25] after the competition.

Three kinds of nodes were used in the implementation: functions, data terminals, and action terminals. To ensure a type-safe conversion of the results data terminals were splitted into numerical and boolean nodes. Several function calls of the API were mapped to terminal nodes and provided the input for the agent's decision making process. In contrast to the work of Alhejali and Lucas we did not include hand-coded action terminals. On the one hand, this will increase the complexity of the evolution process, but on the other hand, we wanted to design an as pure as possible learning process.

A. General Behavior Tree Nodes

Due to the adverserial tasks for both agents we suggest to use a differing set of action and data terminals, which will be discussed in their following respective subsections.

1) Function-Nodes: Function-Nodes are used by both implementations and are listed below:

Control Functions: The main control structure in our behavior is made of If…Then…Else…-nodes and If…Less-Than…-Then…Else…-nodes. The first control node has three children and is evaluating its first subtree for determining which of the other children needs to be evaluated and returned. Our second node type includes four children for which the first two are evaluated and numerically compared. In case the first value is less than the second value the *if*-case will be evaluated, otherwise the control flow will continue with the evaluation of the second tree.

Boolean Functions: In order to simplify the combination of boolean inputs we included nodes for the boolean function *And*, *Or*, *Xor*, and *Not*. Those evaluate their subtrees and apply the appropriate boolean function before returning the results.

RandomNumber: This node generates a random number in the range of $[0, 1]$.

Constants: We also provide constant nodes for the representation of *integer* and *double* numbers, as well as representing the boolean states *true* and *false*. Integer numbers are limited to a range of $[1, 100]$ and were created for comparing the outcome of distance evaluations with fixed thresholds. Double numbers are limited to a range of $[0, 1]$ and will represent probability based decisions in combination with random numbers. We added boolean nodes with fixed values to simplify the mutation process. Therefore, it can quickly turn boolean function evaluators on or off by replacing one of its children with a constant value.

B. PacMan based Genetic Programming Nodes

The Pac-Man controller is based on a single tree with a terminal node output. We decided to use simple methods which wrap specific API calls of the competition framework. It needs to be noted that the partial observation enforced by the games interface forces us to store the availability of unseen pills in internal memory. Queries about the availability and the distance to pills and power-pills are answered with the current information from the game's interface and the internal memory.

1) Data Terminals:

IsPowerPillStillAvailable: Checks if any power pill is still available.

AmICloseToPower: Checks if any power pill is closer than a specified threshold. The threshold can be influenced by mutation and is a fixed value in the range of $[1, 100]$.

IsEmpowered: Checks if Pac-Man is currently able to eat at least one Ghost.

IsGhostClose: Estimates the distance to each Ghost and reports if at least one is closer than a specified threshold. The threshold can be influenced by mutation as discussed above.

SeeingGhosts: Reports if any Ghost is visible in the current state of the game.

DistanceToGhostNr: Sorts the approximate distances to each Ghost and reports the Ghost with the specified rank.

EmpoweredTime: Estimates the empowered time left since the last power pill was eaten.

2) Action Terminals: The following action terminal nodes were included for determining the movement:

FromClosestGhost: Determines the shortest path to the closest Ghost and goes in the opposit direction of the first move. The FromClosestGhost-node provides basic fleeing behavior to the generated Pac-Man controller.

ToClosestEdibleGhost: Moves towards the closest edible Ghost. Non-edible Ghosts are not taken into account.

ToClosestPowerPill: Goes to the closest power pill that is still available.

ToClosestPill: Goes in the direction of the closest pill that is still available.

C. Ghost Team based Behavior Tree Nodes

The Ghost Team is implemented by four separate Ghost controllers, which share information through the restricted messaging protocol. At each game tick Ghosts share their current position and, if in sight, Pac-Man's current position. The information is stored for the next 15 ticks till it becomes updated or deleted. We also check for the power pill availability in each tick. In case it becomes known that a power pill is not available anymore it is removed from an internal list. The game's interface does not allow Ghost controllers to share information about the presence of power pills. So each Ghost needs to explore the current state of a power pill by itself.

1) Data Terminals: Based on this the following data terminal nodes were created for learning Ghost behavior:

SeeingPacMan: Returns if Pac-Man is in the current line of sight.

IsPacManClose: Determines the distance to the last known position of Pac-Man and compares it against a threshold. The threshold can be influences by mutation and is a fixed value in the range of $[1, 100]$.

IsPacManCloseToPower: Determines the distance to the last known position of Pac-Man and all power pills. The smallest value is checked against a threshold. We store the threshold the node itself, so it can be mutated as explained above.

IsEdible: Checks if the Ghost, that is calling this node, is edible.

IsPowerPillAvailable: Checks to the best of this Ghosts knowledge if any power pill is still available for collection.

DistanceToOtherGhosts: Returns the distance to the closest other Ghost.

EstimatedDistanceOptimistic/Pessimistic: Estimates the distance to the last known position of Pac-Man. In case it is outdated an optimistic or pessimistic estimate is returned.

2) Action Terminals: Additionally, to the specialized data terminal nodes we mapped movement specific API calls to action terminal nodes for the Ghosts. The following were created to represent basic chasing and fleeing behavior and return the move to the best of the Ghosts knowledge.

ToPacman & FromPacMan: Returns the first move on the shortest path to the last known position of Pac-Man or away from it.

FromClosestPowerPill & ToClosestPowerPill: Returns the first move on the shortest path to the next available power pill or away from it. Since it is not always known if a power pill still exists it is assumed that, if not seen differently, a pill is still available for collection.

Split & Group: The Split-Node returns a move away from the closest other Ghost. Whereas, the implemented Group-Node returns a move in direction to the closest Ghost.

V. Combining cooperative and adverserial coevolution

To get a better grasp at the performance of each learning process we first validate each genetic programming process on its own. After this, we continue our analysis with a discussion of the results of the combined evolution framework in Section V-B.

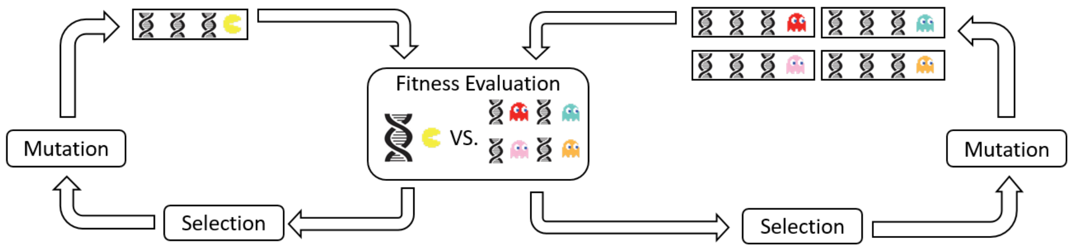


Fig. 3: Evolution process combining cooperative and adverserial tasks for simultaneous generation of both controller types.

A. Single Evolution Process

1) Pac-Man: Similar to the work of Alhejali and Lucas we used genetic programming for evolving a set of Pac-Man agents. The Ghost Team sample implementation provided by the competition's API was used as an opponent. The population size was fixed to 1000 individuals, which were competing for the best average score in 3 games. We used natural selection and kept one-third of the best individuals per generation. New individuals were created using mutation of previously winning individuals.

For each generation we stored the best performing Pac-Man controller, its fitness, and the average fitness of the whole population. The full evoluationary process was repeated ten times. We report the average results and confidence intervals per generation ($\alpha = 0.05$). Our results will be discussed in Section VI-A1.

2) Ghost Team: In the single evolution process we cooperatively evolved four populations of Ghost controllers. We draw one Ghost of each population to form a group of diverse Ghost controllers and play multiple games against hand-coded Pac-Man AIs. Since the game does not provide a score for the Ghost Team, we decided to rate a team by the average fitness value a Pac-Man controller can achieve against them. This process is repeated several times to get a better view on the cooperation between individuals in the separated populations.

For our experiment we chose to use four populations of 250 individuals each. We used natural selection and kept one-third of the best individuals per generation. New individuals were generated using mutations on previous ones, capable of replacing values in single nodes or replacing whole subtrees with random configurations. Due to the enforced return type of each sub-tree, no invalid trees were created during this process. For this work we refrained from implementing additional crossover operators, which may be added at a later point in time.

For each generation we store the best performing team, its fitness, and the average fitness of the generation. Additionally, we repeated this process ten times. We report the average result and confidence intervals per generation ($\alpha = 0.05$). Results will be discussed in Section VI-A2.

We also compared our process for evolving diverse Ghost Teams based on 4 populations of each 250 individuals with the uniform approach of evolving Ghost controllers based on one generation including 1000 individuals. Uniform Ghost Teams include 4 copies of the same Ghost behavior. Results are compared based on their performance in minimizing Pac-Man's score throughout the generations and the confidence intervals per generation.

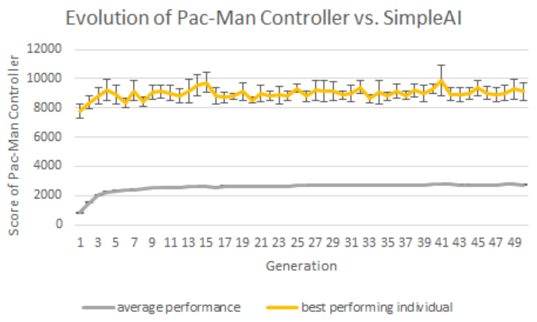


Fig. 4: Performance per generation of evolved Pac-Man controllers. The score is based on the average score each controller achieved in three playthroughs. Additionally the results were averaged over ten repetitions of the experiment. Error bars show the confidence interval for $\alpha = 0.05$.

B. Combined Evolution Process

In contrast to the single evolution processes we tried to evolve both agents in parallel to eliminate the need of creating handmade versions of the concurring player. This also simulates a learning process on both sides. The adversarial tasks of clearing the maze and catching Pac-Man form an adversarial coevolution problem. However, in this special case one part is represented by a cooperative coevolution process. Figure 3 visually represents our evolution process.

During the learning phase we expect to observe several jumps in the fitness value. At those time points, the Pac-Man agent should have learned a strategy to avoid the current generation of Ghost controllers. Over time the Ghost controllers will be able to adapt and, therefore, lower the score achieved by the Pac-Man agent. We stored the best individual of each iteration to provide Ghost controllers on differing skill levels. Both controller types will be evaluated against our handcrafted AIs to prove the increasing quality over the generations. Our results will be discussed in Section VI-B

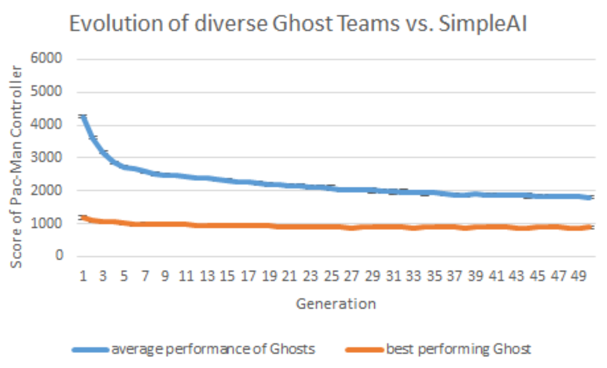


Fig. 5: Performance per generation of evolved Ghost Team controllers. The score is based on the average score a rule-based Pac-Man achieved in three playthroughs and needs to be minimized. Error bars show the confidence interval for $\alpha = 0.05$ but are neglectable.

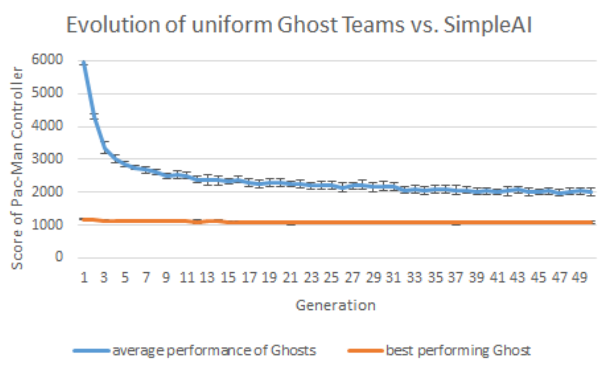


Fig. 7: Performance per generation of evolved Ghost Team controllers using an uniform Ghost Teams. The score is based on the average score a rule-based Pac-Man achieved in three playthroughs and needs to be minimized. Error bars show the confidence interval for $\alpha = 0.05$ but are neglectable.

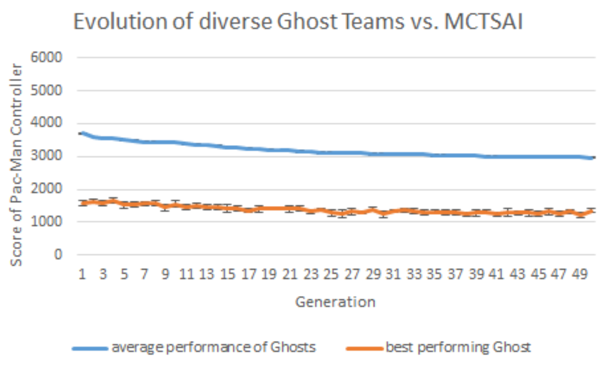


Fig. 6: Performance per generation of evolved Ghost Team controllers. The score is based on the average score a MCTS-based Pac-Man achieved in three playthroughs and needs to be minimized. Error bars show the confidence interval for $\alpha = 0.05$ but are neglectable.

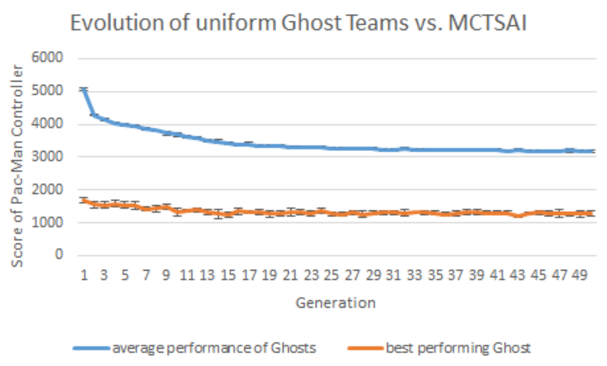


Fig. 8: Performance per generation of evolved Ghost Team controllers using an uniform Ghost Team. The score is based on the average score a MCTS-based Pac-Man achieved in three playthroughs and needs to be minimized. Error bars show the confidence interval for $\alpha = 0.05$ but are neglectable.

VI. Results

A. Single Evolution Process

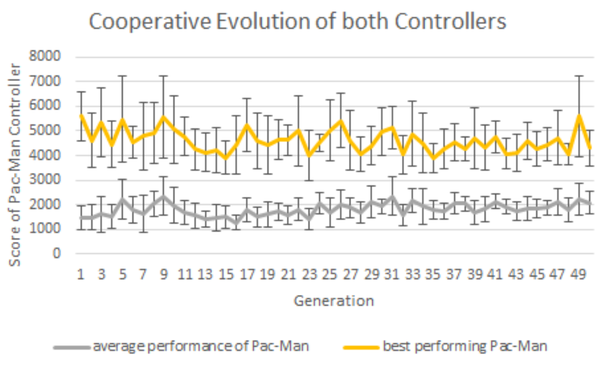
1) Pac-Man Learning Results: Our first test focused on the learning capabilities of our Pac-Man controller. The learning was based on a simple rule-based Ghost controller implementation with shared position tracking of the Pac-Man agent. Problematic was the fitness calculation due to the non-deterministic behavior of the used controllers. However, the individual fitness stabilized over multiple generations. As it is shown in Figure 4 the average performance steadily increased. The performance in the first generations increased much faster than during later generations, which is due to the population converging to more successful strategies.

First successful controllers simply collected the closest pill, which is still available. In the course of generations controllers started to include fleeing behavior. This improved during the following generations which used the provided distance calculations for switching between fleeing and chasing behavior.

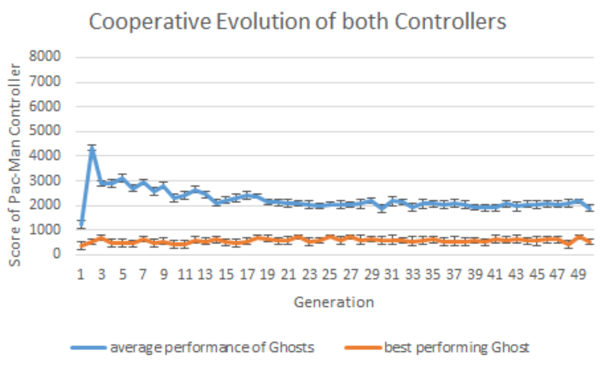
2) Ghost Learning Results: As a next step we evaluated the Ghost learning behavior in two occassions. The first evaluation is based on a simple rule-based Pac-Man implementation. This prefers fleeing from Ghosts, except it is empowered, in which case it will pursue Ghosts for scoring bonus points. In case Pac-Man is currently not endangered he goes for the nearest available pill or power pill. Figure 5 shows our learning results.

The behavior of our Ghosts are still very limited. Depending on the current game state they either split or group up and defend the nearest power pill. In case Pac-Man comes to collect it, they will start to chase him. Those simple rules were enough to decrease the number of points Pac-Man is able to score.

However, with our second evaluation we let the Ghost controllers train against a MCTS-based Pac-Man implementation and hoped for much more elaborate strategies. The fitness results per generation are shown in Figure 6. The average fitness improved much slower due to the stronger play of the opponent. Nevertheless, in contrast to the simple AI the created agents were much more elaborate in their counterplay. The



(a) Evolution of Pac-Man Controllers



(b) Evolution of Ghost Controllers

Fig. 9: Performance per generation of evolved controllers. The score is based on the average score an individual achieved against the best performing opponent of the previous generation. Error bars show the confidence interval for $\alpha = 0.05$.

winning team mixed splitting, grouping, chasing, and defending power pills depending on the game state.

Furthermore, we checked if a team of diverse Ghosts performs better than a team of uniform Ghosts. Therefore, we repeated the experiments with just one population of 1000 Ghosts and created teams by multiple instances of the same Ghost. The performance results of each generation is shown in Figures 7 and 8. While the final performance is approximately the same, the convergence is much slower. First generations of uniform teams performed worse than teams consisting of diverse Ghost controllers.

B. Combined Evolution Process

Our final evaluation covers the coevolution of both agents. Due to the faster convergence of diverse Ghost Teams, we decided to use those for an increased dynamic between both contesting parties. The average result from 10 runs is shown in Figure 9. To get a better view on the development Figure 10 present one single run of the evolutionary process. As expected, the fitness curves of Pac-Man as well as the Ghost Team show several bumps. Reviewing the Pac-Man agents of each generation showed that for example a strong improvement of the best Pac-Man from generation four to generation five is caused by an additional check if Pac-Man is currently empowered. In case he is, he stops eating the closest pill and starts pursuing Ghosts. Ghosts of the next generation learned to flee from Pac-Man on several occassions. In course of the following generations this avoidance behavior was established in most of the populations, which explains the steady decline in the average performance of Pac-Man controllers.

Such adaptations repeated in the upcoming generations. After most of the Ghosts learned to flee from Pac-Man, Pac-Man controllers established a more passive play style. This led to more aggressive Ghosts, which in turn were later countered by more aggressive Pac-Man behaviors. We observed that those changes cycle and after multiple generations similar behaviors established repeatedly. This is also represented by the generally smaller trees in the combined evolution process.

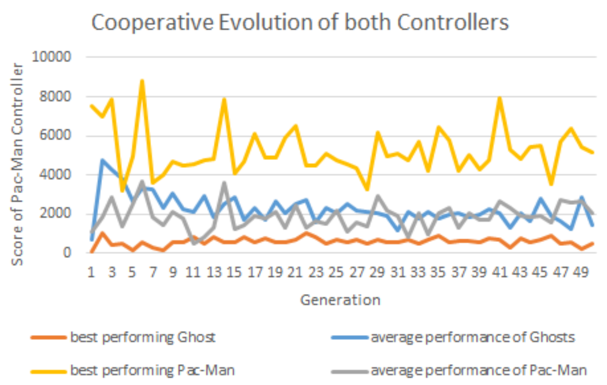


Fig. 10: Detailed illustration of a single run of our coevulationary process.

VII. Conclusions

While the learning outcome of each single evaluation had very promising results, the combined coevolution did not led to high level play. Both, Pac-Man and Ghost controllers were successful in evolving complex strategies for countering the behavior of their current opponent. In repeated games against a non-changing player we were able to develop agents with strong counterplay.

The combined coevolution adapted both controller types simultaneously. Since each Pac-Man base-strategy (flee, pursue, collect) was countered by another Ghost base-strategy (group, split, pursue), the agents had no need to advance the strategies themself. Therefore, high level play evolved very slowly and is not comparable with the outcome of each single learning process. Reducing the selection pressure or preserving agents for multiple generations might help in expanding strategies to the current enemy behavior. Furthermore, increasing the capabilities of the mutation operator or developing a crossover operator might help in increasing the adaptation speed.

Our approach proved to be useful to create a diverse set of agents for both player types. Each generated agent, regardless of which it was created in the single or combined evolution process, can be used for playing the game against any other (human/digital) player. Splitting the Ghost agents in four populations lead to diverse behaviors which complemented each other. During the course of early generations each population converged to one behavior type. Further generations lead to improvements of all agents in their respective fields. This two-phase development reflects exploration and exploitation of possible playing strategies in each population.

In respect to previous studies by Alhejali and Lucas [20] as well as Brandstetter and Ahmadi [21] we will continue our analysis based on changing the complexity of available nodes, which might help in reducing the complexity of the learning process. This can lead to smoother transitions between winning strategies and increase the total complexity of resulting trees.

References

[1] M. Campbell, a. J. Hoane Jr., and F.-h. Hsu, "Deep Blue," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] N. Brown, C. Kroer, and T. Sandholm, "Dynamic thresholding and pruning for regret minimization," 2017.

[4] J. P. A. M. Nijssen and M. H. M. Winands, *Playout Search for Monte-Carlo Tree Search in Multi-player Games*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 72–83.

[5] F. van Lishout, G. M. J.-B. Chaslot, and J. W. H. M. Uiterwijk, "Monte-Carlo Tree Search in Backgammon," *Proc. Comput. Games Workshop*, pp. 175–184, 2007.

[6] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, jul 2011.

[7] P. R. Williams, D. Perez-Liebana, and S. M. Lucas, "Ms. Pac-Man Versus Ghost Team CIG 2016 competition," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, sep 2016, pp. 1–8.

[8] P. Rohlfshagen and S. M. Lucas, "Ms Pac-Man versus Ghost Team CEC 2011 competition," in *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, jun 2011, pp. 70–77.

[9] P. R. Williams, "Ms. Pac-Man Vs. Ghost Team Competition 2017," http://www.pacmanvghosts.co.uk.

[10] M. Gallagher and a. Ryan, "Learning to play pac-man: An evolutionary rule based-approach," *in Proc*, vol. 03, pp. 2462–2469, 2003.

[11] B. K. B. Tong and C. W. Sung, "A Monte-Carlo approach for ghost avoidance in the Ms. Pac-Man game," *2nd International IEEE Consumer Electronic Society Games Innovation Conference, ICE-GIC 2010*, 2010.

[12] D. Robles and S. M. Lucas, "A simple tree search method for playing Ms. Pac-Man," *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pp. 249–255, 2009.

[13] S. Samothrakis, D. Robles, and S. M. Lucas, "Fast Approximate Max-n Monte-Carlo Tree Search for Ms Pac-Man," *IEEE Trans. Comp. Intell. AI Games*, vol. 3, no. 2, pp. 142–154, 2011.

[14] K. Q. Nguyen and R. Thawonmas, "Applying Monte-Carlo Tree Search to collaboratively controlling of a Ghost Team in Ms Pac-Man," *2011 IEEE International Games Innovation Conference, IGIC 2011*, pp. 8–11, 2011.

[15] N. Ikehata and T. Ito, "Monte-Carlo tree search in Ms. Pac-Man," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. IEEE, aug 2011, pp. 39–46.

[16] S. Lucas, "Evolving a neural network location evaluator to play ms. pac-man," *IEEE Symposium on Computational Intelligence and . . .*, pp. 203–210, 2005.

[17] M. Gallagher and M. Ledwich, "Evolving pac-man players: Can we learn from raw input?" *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games, CIG 2007*, no. Cig, pp. 282–287, 2007.

[18] R. Kruse, C. Borgelt, C. Braune, S. Mostaghim, and M. Steinbrecher, *Computational Intelligence*, 2nd ed., ser. Texts in Computer Science. London: Springer London, 2016. [Online]. Available: http://link.springer.com/10.1007/978-1-4471-5013-8http://link.springer.com/10.1007/978-1-4471-7296-3

[19] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[20] A. M. Alhejali and S. M. Lucas, "Evolving diverse Ms. Pac-Man playing agents using genetic programming," *2010 UK Workshop on Computational Intelligence, UKCI 2010*, 2010.

[21] M. F. Brandstetter and S. Ahmadi, "Reactive control of Ms. Pac Man using information retrieval based on Genetic Programming," *2012 IEEE Conference on Computational Intelligence and Games, CIG 2012*, pp. 250–256, 2012.

[22] R. P. Wiegand, "An Analysis of Cooperative Coevolutionary Algorithms," Ph.D. dissertation, GeorgeMason University, 2003.

[23] A. B. Cardona, J. Togelius, and M. J. Nelson, "Competitive coevolution in Ms. Pac-Man," *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pp. 1403–1410, 2013.

[24] "Cooperative and Adverserial Genetic Programming Implementation for the Ms. Pac-Man vs. Ghost Team Competition," http://fuzzy.cs.ovgu.de/wiki/pmwiki.php/Mitarbeiter/Dockhorn?userlang=en.

[25] "Cooperative and Adverserial Genetic Programming Implementation for the Ms. Pac-Man vs. Ghost Team Competition," http://fuzzy.cs.ovgu.de /wiki/pmwiki.php/Mitarbeiter/Dockhorn?userlang=en.