FRIGHT: A Flexible Rule-Based Intelligent Ghost Team
for Ms. Pac-Man

David J. Gagne and Clare Bates Congdon, Senior Member, IEEE

Abstract—FRIGHT is a rule-based intelligent agent for playing the ghost team in the Ms. Pac-Man vs Ghosts Competition held at the 2012 IEEE Conference on Computational Intelligence and Games. FRIGHT uses rule sets with high-level abstractions of the game state and actions, and employs evolutionary computation to learn rule sets; a distributed homogenous-agent approach is used. We compare the performance of a hand-coded rule set to one learned by the system and find that the rule set learned by the system outperforms the hand-coded rules.

Keywords—rule-based system, evolutionary computation, games, Ms. Pac-Man

I. INTRODUCTION

VIDEO games provide excellent test beds for artificial intelligence (AI) approaches because they offer real-time interactive environments in which an approach may be evaluated without the external factors inherent in real-world environments. Standing in stark contrast to the board games employed in early AI research, classic arcade games feature relatively simple controls (such as a joystick) and require quick reactions from the player. In Pac-Man, one of the best known arcade games, the player guides the Pac-Man character through a 2D maze and must score points by “eating” pills while avoiding capture by a team of four ghosts. In such a fast-paced environment, a game-playing agent, like a human player, must interpret its environment and make decisions in a fraction of a second, without the benefit of extensive planning. The Ms. Pac-Man game is very similar to Pac-Man, but, unlike its predecessor, it is nondeterministic; there is no fixed sequence of moves by which a player will always win. Its simple interface, rapid pace, and stochastic nature make Ms. Pac-Man a superb environment for evaluating intelligent artificial agents.

Recently, conferences on artificial intelligence have begun to include competitions for creating game-playing agents. The Ms. Pac-Man AI Competition, first held at the 2007 IEEE Conference on Evolutionary Computation (CEC), allows participants to submit agents for playing a simulation of the original Ms. Pac-Man game [1]. The Ms. Pac-Man vs. Ghosts Competition, which was first held at the 2011 IEEE Conference on Computational Intelligence and Games (CIG), lets participants submit artificial agents for the Ms. Pac-Man character or for her adversaries, the team of four ghosts [2]. Ms. Pac-Man agents are played against the ghost teams in a round-robin style tournament, and the score attained by Ms. Pac-Man is recorded for each game. The Ms. Pac-Man agent with the highest average score is declared the winner, while the ghost team with the lowest average score is the winner. Even though the focus of the conference is on computational intelligence, agents based on any algorithm or approach, including hand-coded agents, are allowed to compete.

The task of controlling the ghost team in Ms. Pac-Man could be handled by a single agent that observes the game environment and assigns moves to the individual ghosts. Such a centralized system for controlling the ghost team would be feasible to design and quite possibly effective for the Ms. Pac-Man game. In real-world environments, such as search and rescue, reliance on a centralized controller becomes a liability. It is difficult to design a centralized controller capable of handling every contingency, whereas a distributed system is more robust in an unpredictable environment [3]. In a distributed system, each agent on a team acts independently, even though its decisions may be influenced by other agents. Since the ghosts can more readily capture Ms. Pac-Man by working together, we chose the task of developing a Ms. Pac-Man ghost team as a test bed for our approach to developing coordinated multi-agent teams.

In this paper, we present a system for developing a ghost team for Ms. Pac-Man. We call the system FRIGHT; which stands for Flexible Rule-based Intelligent Ghost Team. In our system, each agent uses a rule set to select its behavior based on high-level abstractions of the game environment. All agents on the team use the same rule set, though each decides independently what its next action will be. Thus, this is a distributed approach with homogeneous agents. In this work, we apply evolutionary computation (EC) to evolve rule sets to be used by the ghost agents and compare the learned rules to a hand-coded rule set. We plan on entering a ghost team controlled by FRIGHT agents into the Ms. Pac-Man vs. Ghosts Competition held at the 2012 CIG conference.

The remainder of this paper proceeds as follows: Section II describes the task and related work; Section III describes the design of FRIGHT, including the representation of the game environment; Section IV describes the learning mechanism employed by FRIGHT; Section V describes the experiments we ran; Section VI presents the results of those experiments; in Section VII, we draw some conclusions from the results; and Section VIII describes future work.

II. BACKGROUND

This section describes the game used in the Ms. Pac-Man vs. Ghosts competition and related work.
A. Task Overview

The Ms. Pac-Man vs Ghosts Competition uses a simulated version of the Ms. Pac-Man video game. While the simulation retains many aspects of the original arcade game, other aspects have modified for the competition. The goal of the Ms. Pac-Man agent remains the same as the goal for a human player: to score as many points as possible before running out of lives. Ms. Pac-Man starts a game with three lives, and she loses a life each time she is captured (touched) by a ghost. She is awarded an additional life when the score reaches 10,000 points. The game is played in a series of four mazes (levels), and Ms. Pac-Man scores points by “eating” three different types of objects:

- **Pills**: Each maze contains numerous dots, or pills, which are each worth 10 points.
- **Power Pills**: Each maze contains four power pills near the corners of the maze, which are worth 50 points each. In addition, these power pills turn the ghosts edible for a short time.
- **Edible Ghosts**: When Ms. Pac-Man consumes a power pill, all four ghosts become edible for a short interval. The first edible ghost Ms. Pac-Man consumes during the interval is worth 200 points, the second is worth 400, the third 800, and the fourth 1600, for a potential total of 3000 points. When the ghosts are edible, they also move at reduced speed.

The goal of the ghost team is to minimize Ms. Pac-Man’s score. At the start of each level and after Ms. Pac-Man is captured, the ghosts are released one at a time from a cage in the center of the maze. Whereas Ms. Pac-Man is permitted to move in any direction she chooses, the ghosts cannot reverse direction and may only change direction upon reaching a junction. Even though the ghosts move at the same speed as Ms. Pac-Man and outnumber her, their inability to reverse direction increases the difficulty of developing a strategy for the ghost team. In addition, if Ms. Pac-Man survives a level for 2 minutes, she is awarded half of the points she would receive for eating the remaining pills. This reduces the effectiveness of defensive ghost team strategies. Furthermore, at random intervals throughout the game, global reversal events occur in which all ghosts reverse direction. These events add a layer of unpredictability to the game, even if the agents for both sides use deterministic algorithms.

When Ms. Pac-Man clears a level by consuming all of the pills and power pills in a maze, the game play resumes in the next maze with the ghosts and Ms. Pac-Man in their respective starting positions. The four mazes in the game are played in a cycle; if Ms. Pac-Man clears the fourth maze, the next level of the game uses the first maze (this sequence of the mazes differs from the original arcade game). With each advance in level, the duration of a power pill’s effects decreases; that is, the ghosts remain edible for a shorter period of time. In addition, the time during which the ghosts remain in the cage at the start of a level and after capturing Ms. Pac-Man decreases as the levels advance, increasing the difficulty for Ms. Pac-Man as the game progresses.

B. Related Work

Games have been used in artificial intelligence and machine learning research since the 1950's, when board games such as checkers and chess were the focus [4]. In recent years, focus has expanded to include a variety of video games, such as Ms. Pac-Man. Several learning and search techniques have been applied to developing agents for Ms. Pac-Man, including: rule-based systems [5], genetic programming [6], artificial neural networks [7], and Monte-Carlo tree search (MCTS) [8].

Rule-based agents have been used to create successful agents for a variety of games, including Ms. Pac-Man. A rule-based agent uses a set of if-then rules to select its actions based on conditions observed in its environment. Gallagher and Ryan [9] use a rule-based approach and population-based incremental learning to develop an agent that plays a simplified version of Ms. Pac-Man (with only a single ghost). Sitzá and Lórinz [10] use an optimization technique known as the cross-entropy method to learn low-complexity rules for playing Ms. Pac-Man. Fitzgerald and Congdon [5] describe a rule-based agent for playing Ms. Pac-Man (RAMP) that won the 2008 Ms. Pac-Man AI Competition at the IEEE World Conference on Computational Intelligence (WCCI) [11]. RAMP uses high-level abstractions of the environment as conditions and complex behaviors as actions. Building upon the success of the RAMP agent, Small and Congdon [12] developed Agent Smith, a rule-based agent that plays the first-person shooter game Unreal Tournament 2004 and uses evolutionary computation to improve its rule sets. The REALM agent developed by Bojarski and Congdon [13] won the Mario Learning Competition at CIG 2010 [14]. REALM uses EC to evolve sets of rules with high-level conditions and actions.

While relatively little research has focused on developing a team of ghost agents for playing Ms. Pac-Man, there has been extensive work in applying learning techniques to the multi-agent problem; Panait and Luke provide a review of work in cooperative multi-agent learning in [15]. Wittkamp, Barone, and Hingston [16] use the NEAT approach [17] to evolve neural networks for controlling the ghosts in
Pac-Man. Beume et al. [18] compare strategies learned by neural networks to those learned in low-level rules in a Ms. Pac-Man clone and find that both approaches show improvement over time. Yannakakis and Hallam [19] use evolutionary computation with neural networks to develop ghost agents that learn to adapt to Ms. Pac-Man’s strategy during game-play, but the focus of their work is to produce more interesting opponents, rather than the most efficient ghost team. In this project, we extend the work of [5], [12], and [13] to the multi-agent problem of developing a team of ghost agents in a simulation of Ms. Pac-Man.

III. SYSTEM DESIGN

Each FRIGHT agent employs a rule-based system to choose its next move based on the state of the game. Like RAMP, Agent Smith, and REALM, a FRIGHT agent’s rule set uses high-level abstractions of the game as conditions and actions. Since the rule set works at a high level, the agent uses an internal representation of the game to translate from the game state to conditions and from the action selected by the rule-based system to the agent’s next move.

A FRIGHT agent uses the following steps to choose its next move based on the game information provided by the Ms. Pac-Man vs Ghost simulation:

1) Translate the game state into high-level conditions.
2) Find rules for which all conditions have been met.
3) Choose one of the rules to fire.
4) Translate the action specified by the rule into the next move made by the agent.

In Section III-A, we describe the internal representation of the game used by each agent. In Section III-B, we describe the vocabulary (the conditions and actions) of the rules used by FRIGHT and the method by which the rule-based system chooses a single rule to fire. In Section III-C, we describe a set of hand-coded rules constructed using the vocabulary, which provide us with a basis for comparison when we examine the rule sets learned by evolutionary computation.

A. Representation of the Game Environment

Since FRIGHT uses high-level abstractions of the game state as conditions to each rule and high-level behaviors as the resulting actions, an agent must rapidly translate the state of the game into conditions and to translate the action selected by the rule-based system into the basic moves the agent may take (up, down, left, or right). To facilitate this translation, a FRIGHT agent uses an internal representation of the maze as a graph (see Figure 1); a similar approach was used by RAMP to represent the game. A corridor in the maze (an area of the maze where movement is restricted by the walls to two possible choices) is represented as pair of weighted directed edges in the agent’s graph, one for each direction of travel along the corridor. Each intersection in the maze (where multiple corridors meet) is represented as a node in the agent’s internal graph. Each edge is assigned a weight based on the number of times steps required to traverse the edge.

At each time step, the agent uses information from the Ms. Pac-Man vs. Ghosts API to update its position in the internal graph. Since a ghost is not permitted to reverse direction, a ghost moving along a corridor has no choice but to continue advancing along the corridor. Because of this, a ghost agent only needs to make a decision when it reaches an intersection. When this occurs, the agent also estimates the positions of the other ghosts and of Ms. Pac-Man on its internal graph. To simplify the translation of the game state to rule conditions and of the actions to moves, the positions of the ghosts and of Ms. Pac-Man in the agent’s internal graph are approximated as the nearest intersection to the entity.

When a decision is required of a FRIGHT agent, its internal representation of the environment is used to assign numeric values to fifteen conditions describing the state of the game; these are described in Table 1.

The abstractions of the game state were selected based on observation of the game and represent the factors a human player might consider when deciding the next move. The amount of time remaining in the game, the count of remaining power pills, and the count of remaining regular
pills are provided directly by the Ms. Pac-Man vs. Ghosts simulation. The maze number is obtained by taking the base four modulo of the current level, which is available from the game API. The Edible condition is set to 2 if the agent will remain edible for more than 100 time steps, 1 if the remaining edible time is between one and 100 steps, and 0 if the agent is not edible. The conditions measuring proximity (Pill Prox, Engaged, etc.) are assigned a value from 0 to 2, based upon whether the agent is “not near” (0), “near” (1), or “very near” (2) an entity (See Figure 2). Two entities are considered very near one another if the shortest path between the entities contains no more than one traversable edge. Entities are considered near one another if the shortest path between the entities contains no more than two traversable edges. Since the agent is not permitted to reverse direction, the edge by which the agent has reached its current node (the reverse edge) is not considered when determining the distance from the agent to another entity. Since Ms. Pac-Man is allowed to traverse edges that the agent may not, the Engaged condition is not symmetric with the MPM Prox condition. The Allies Between condition is assessed by finding the shortest non-reversing path from the agent to Ms. Pac-Man; for each ghost agent occupying a node along this path, the value of this condition is increased by one.

### B. Rules

In FRIGHT, each rule consists of fifteen conditions that must be matched by the game state before the rule can be selected by the system and a single action that is taken if the rule is selected (fires). As mentioned previously, all four ghosts use the same rule set.

1) Conditions: Each rule in FRIGHT has fifteen conditions that correspond to the game state conditions shown in Section III-A. Each rule condition specifies a numeric range (a minimum and maximum value) or a “Don’t Care” value, which means that the condition is considered satisfied for any value of the game state. If the corresponding game state value falls within the range given by the rule condition, then that condition is considered satisfied. For example, if a rule specifies a range of 2-4 for Power Pills, then the rule will never fire when only one power pill remains. The Don’t Care value permits rules that ignore the state of a condition or several conditions entirely, allowing rules that depend on only a few conditions (or even default rules that fire for any game state). A rule with fewer Don’t Care valued conditions is said to be more specific than a rule with more Don’t Care values, since its conditions are satisfied over a narrower range of game states.

2) Actions: Each FRIGHT rule specifies a single action to be taken when the rule fires. Each of the six actions in the FRIGHT vocabulary determines a high-level behavior for the agent. Each action specifies a target or set of targets; when an action is selected by the rule-based system, the agent moves toward the target. If a set of targets is specified, then the agent moves toward the closest member of the target set. If the agent is at the target node, then the next-nearest potential target is selected. Targets may include Ms. Pac-Man, uneaten power pills, and intersections where four corridors meet (“hubs”). In addition to a target, each action includes a set of entities that the agent should avoid, along with a priority level for avoiding the entity. For example, there are situations (i.e. when the agent is edible) when the agent should avoid Ms. Pac-Man.

The six actions used by FRIGHT are: Retreat, Evade, Surround, Attack, Protect, and Defend. An agent in Attack mode will take the shortest path to Ms. Pac-Man. In Surround mode, an agent will also target Ms. Pac-Man, but it will avoid other agents. Agents in Surround mode will spread out, closing off more of Ms. Pac-Man’s potential escape routes than would ghosts in a cluster. The Retreat action sends an agent to the nearest power pill while avoiding Ms. Pac-Man and other ghosts. Avoiding other ghosts will reduce the opportunities for Ms. Pac-Man to eat high-scoring clusters of edible ghosts. The Protect action is similar to Retreat, but it does not induce the agent to avoid Ms. Pac-Man. The Evade action sends an agent toward the nearest hub while avoiding Ms. Pac-Man and other ghosts, while the Defend

<table>
<thead>
<tr>
<th>Condition</th>
<th>Represents</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edible</td>
<td>Agent’s edible status</td>
<td>0-2</td>
</tr>
<tr>
<td>Pill Prox</td>
<td>Agent’s proximity to a power pill</td>
<td>0-2</td>
</tr>
<tr>
<td>Engaged</td>
<td>Agent’s proximity to Ms. Pac-Man</td>
<td>0-2</td>
</tr>
<tr>
<td>MPM Prox</td>
<td>Ms. Pac-Man’s proximity to the agent</td>
<td>0-2</td>
</tr>
<tr>
<td>MPM Pill Prox</td>
<td>Ms. Pac-Man’s proximity to power pill</td>
<td>0-2</td>
</tr>
<tr>
<td>Allies Very Near</td>
<td>Count of allies (other ghosts) very near the agent</td>
<td>0-3</td>
</tr>
<tr>
<td>Allies Near</td>
<td>Count of allies near the agent</td>
<td>0-3</td>
</tr>
<tr>
<td>Allies Engaged</td>
<td>Count of allies very near Ms. Pac-Man</td>
<td>0-3</td>
</tr>
<tr>
<td>Allies Closing</td>
<td>Count of allies near Ms. Pac-Man</td>
<td>0-3</td>
</tr>
<tr>
<td>Allies Between</td>
<td>Count of allies between agent and Ms. Pac-Man</td>
<td>0-3</td>
</tr>
<tr>
<td>Power Pills</td>
<td>Count of the power pills remaining</td>
<td>0-4</td>
</tr>
<tr>
<td>Pills</td>
<td>Count of the regular pills remaining</td>
<td>0-174</td>
</tr>
<tr>
<td>Escapes</td>
<td>The degree of the intersection nearest Ms. Pac-Man</td>
<td>2-4</td>
</tr>
<tr>
<td>Maze</td>
<td>The maze currently being played</td>
<td>1-4</td>
</tr>
<tr>
<td>Time</td>
<td>Time remaining in the current level</td>
<td>1-3000</td>
</tr>
</tbody>
</table>
TABLE 2
THE WEIGHTS APPLIED TO GRAPH EDGES FOR EACH ACTION

<table>
<thead>
<tr>
<th>Action</th>
<th>Targets</th>
<th>Avoids (high)</th>
<th>Avoids (medium)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack</td>
<td>Ms. Pac-Man</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Surround</td>
<td>Ms. Pac-Man</td>
<td>Allies</td>
<td>–</td>
</tr>
<tr>
<td>Retreat</td>
<td>Power pill</td>
<td>Ms. Pac-Man</td>
<td>Allies</td>
</tr>
<tr>
<td>Protect</td>
<td>Power pill</td>
<td>–</td>
<td>Allies</td>
</tr>
<tr>
<td>Evade</td>
<td>Hub</td>
<td>Ms. Pac-Man</td>
<td>Allies</td>
</tr>
<tr>
<td>Defend</td>
<td>Hub</td>
<td>–</td>
<td>Allies</td>
</tr>
</tbody>
</table>

Fig. 3. Weights are added to edges of the graph to elicit avoidance behavior from the FRIGHT agent. In the figure, the orange ghost has selected the Retreat action. Weights are added to the edges approaching Ms. Pac-Man or the ally.

3) Action Selection: When the agent receives the game state from the simulation, it searches its rule set for a rule whose conditions have all been satisfied. If there are no rules for which all conditions are satisfied, then the default action for the agent is to Attack Ms. Pac-Man. If a single rule’s conditions have been met, then the agent takes the action specified by the rule. If the conditions for more than one rule have been satisfied, then the most specific rule (the rule with the fewest Don’t Care values) that matches all of the game state conditions is fired. If there are multiple rules with all conditions satisfied and if the rules contain an equal count of Don’t Care conditions, then the rule that occurs earliest in the rule set is selected to fire.

4) Using the Maze to Resolve Actions: When resolving an action, the agent uses Dijkstra’s single-source shortest paths algorithm to find the shortest path from the agent to the target. Applying a very large weight (1,000 distance units) to the reverse edge before the shortest path is calculated ensures that the path selected by the agent is non-reversing. In order to encourage avoidance behavior, a weight is applied to any edges leading into the entity being avoided, and a lesser weight is applied to the edges leading to nodes adjacent to the entity being avoided (See Table 2 and Figure 3). For example, if the action specifies avoiding Ms. Pac-Man with “high” priority, then any edges leading into Ms. Pac-Man’s current node receive a “high” penalty (50), and any edges leading into nodes adjacent to Ms. Pac-Man are given a “medium” penalty (25).

C. Hand-coded Rules

FRIGHT is capable of loading rule sets from text files. This allows the system to store successful rule sets at the end of a learning run for future use; it also permits the user to design and store a rule system based on observation of the game. Before we applied evolutionary computation to the problem of evolving rule sets for FRIGHT, we designed a “hand-coded” rule set for the system. Developing a rule set “by hand” allowed us to study the rule-based system and observe it in action before commencing any learning runs. This rule set also serves as a basis of comparison with the evolved rule sets. In future work, the hand-coded rules may be used to seed the initial population of a run of the evolutionary algorithm used for learning.

The hand-coded set includes six rules (See Table 3). The first rule instructs an edible agent to retreat to the nearest active power pill while avoiding Ms. Pac-Man (and with lesser priority, other ghosts). If no power pills remain, then the second rule instructs the agent to flee toward the nearest hub. The next rule instructs the agent to attack Ms. Pac-Man if the agent is very close to her. If Ms. Pac-Man is at an intersection with only three escape routes and there are three other agents very near Ms. Pac-Man, then the fourth and fifth rules instruct an agent who is not nearby to guard either a power pill or a hub, depending on whether any power pills remain in the maze. The final rule instructs the agent to Surround Ms. Pac-Man.

IV. LEARNING

To implement the learning of rule sets in FRIGHT, we use the Java-based evolutionary computation system ECJ [20]. ECJ includes packages for several styles of evolutionary computation, including evolutionary strategies (ES). We use the simple evolution procedure provided by ECJ; the evolution procedure follows the following pattern:

1) Generate an initial population of rule sets (at random).
2) Evaluate rule sets.
3) Breed new rule sets from select population members.
4) Repeat 2-3 for a specified number of generations.

A. Evaluation Phase

To evaluate a rule set, FRIGHT creates a team of four identical agents using the rule set. The team plays some fixed number of games of the Ms. Pac-Man vs Ghosts simulation against the Starter Ms. Pac-Man agent included with the API, which exhibits the following behaviors (in order of precedence):

- If a non-edible ghost is nearby, move away.
- Eat the nearest edible ghost.
- Eat the nearest pill or power pill.

Once the pre-determined number of games has been played between the ghost agents and the Starter Ms. Pac-Man, the average score of the series of games is subtracted from 100,000 to yield the fitness score for the rule set. This is done because the simple evolution procedure in ECJ is configured by default to optimize an increasing fitness function.
B. Breeding Phase

The breeding phase in a FRIGHT evolutionary run uses a $\mu + \lambda$ breeding strategy. In a $\mu + \lambda$ strategy, the $\mu$ best individuals of the population are chosen after the evaluation phase. The $\mu$ parents are used to produce $\lambda$ children, with each parent producing $\lambda / \mu$ children. In ECJ, $\lambda$ is constrained to be a multiple of $\mu$. The parents are retained into the next generation, so the total size of the population after the breeding phase is always $\mu + \lambda$. This strategy was used by REALM to successfully learn rule sets for a Mario agent.

In the breeding phase, the child rule sets are allowed to vary slightly from the parents through genetic operators. ECJ includes a package for evolving rule sets that has basic operators that are applied with certain probability. In rule crossover, a rule from one of the child sets is swapped with a rule from another child set. The mutation operator changes the rule conditions and actions; each condition and action within a rule has some probability of being mutated. When an action is mutated, a new action is selected at random from all possible FRIGHT actions to replace the old value (because the old value is not excluded from selection, there is some chance that the action does not change as a result of the mutation). When a condition is selected for mutation, it becomes a Don’t Care condition with some probability. Otherwise, a pair of values within the range allowed for the condition is chosen at random. A Don’t Care condition selected for mutation will always be changed to a numeric-valued condition.

V. Methodology

We conducted experiments in two phases: a learning phase and a comparison phase. In the learning phase, the ES was run for 500 generations using the parameters described below. In the comparison phase, we measured the performance of the hand-coded rule set, the performance of most fit rule set of the initial population used in learning (which was generated at random), the performance of the best rule set found in 500 generations of evolution, and the performance of the Aggressive Ghost Team included with the Ms. Pac-Man vs. Ghosts API. In this controller, the ghosts always attack. Each rule set was used by FRIGHT in 10,000 games against the Starter Ms. Pac-Man controller. The same parameters for the FRIGHT conditions and actions were used for all rule sets in both the learning phase and the comparison phase of the experiments.

A. Learning Parameters

We used 10 ES runs with different seeds to the random number generator in the learning phase, but only rule sets from the most successful learning run (the run with the largest increase in fitness over 500 generations) were used in the comparison phase. At the start of an ES run, an initial population of 105 rule sets (each consisting of 20 rules) was generated at random, with each condition given a 40% probability of being assigned a Don’t Care value. A series of 100 games against the Starter Ms. Pac-Man was used to evaluate each rule set. The best $\mu = 5$ rule sets were selected from each generation to become parents. The parent rule sets were used to generate $\lambda = 100$ children. Each child rule set was subject to rule crossover with a 10% probability. Operators for varying rule set length were not employed, due to a bug in the version of ECJ available at the time (version 19). Each condition and action had a 10% probability of mutation, and the probability of a numeric condition becoming a Don’t Care condition was 40%. The evolutionary algorithm was stopped on the $501^{st}$ generation, and the best rule set of the entire run was used in the comparison phase, along with the best rule set of the initial population.

VI. Results

The lowest score of each generation is shown in Figure 4 for the learning run that resulted in the lowest scoring team of all 10 learning runs. The best rule set of the initial population for this run allowed an average of 4,968 points in 100 games against the Starter Ms. Pac-Man, while the best rule set learned after 500 generations allowed an average score of 3,732 in 100 games, which is a decrease of 1,235
The best rule set learned by FRIGHT (the “Evolved Rule Set”) is shown in Table 3. Its first rule instructs non-edible agents in the first or second maze to surround Ms. Pac-Man unless she is very near a power pill. The ghost team using the evolved rule set allowed an average of 4,788 points on agents with hand-coded rule sets allowed 4,552 points in the first generation to the last, and the average score decrease from start to finish over all 10 learning runs was 833 points.

The standard deviation for the population of 10,000 games against the Starter Ms. Pac-Man was 2,067 for the evolved rule set, 2,309 for the hand-coded rule set, and 2,512 for the first generation rule set, and the aggressive ghosts averaged 5,878 points (see Figure 5). The standard deviation for the population of 10,000 games in the first generation rule set averaged 5,515 points, or 25% of the first generation score. The worst of the learning runs showed a reduction of only 180 points from the first generation to the last, and the average score decrease for improving the system remain.

Perhaps the optimal rule set had been found, but based on the strange rules appearing in the final rule set, it seems more likely that we have not found the best parameters for scoring rule set across the 10 runs after 500 generations. These results demonstrate that EC can be used to learn rule sets for FRIGHT agents that produce a more successful ghost team than some hand-coded rule sets, but opportunities for improving the system remain.

The best rule set after 500 generations of the ES achieved a lower average score than both the hand-coded and randomly generated rule sets, but the best score for this run was achieved by the 179th generation. Table 3 shows the highest-scoring rule set across the 10 runs after 500 generations.

TABLE 3
The hand-coded rule set and the highest-scoring rule set after 500 generations. Empty cells represent a Don’t Care condition. Cells in gray indicate an effective Don’t Care condition.
the ES should allow rule sets of varying size. We also find that several rules, including the first, evolved numeric conditions that allowed the entire range of values for the game state, producing effective Don’t Care conditions; however, since the action selection mechanism gives precedence to more specific rules, effective Don’t Care conditions do not decrease a rule’s likelihood of being selected.

The scores of the games were noisy. The FRIGHT team using the learned rule set allowed as many as 11,620 points and as few as 950 points. While some noise is to be expected, the large variation observed in the scores suggests that the conditions and actions need to be refined. For instance, an agent targets the nearest power pill when the Retreat action is employed, but a safer target would be the power pill furthest from Ms. Pac-Man. In addition, the learning may not have been allowed to continue long enough; longer learning times could help further refine the ghost team’s strategy.

Even though the FRIGHT agents learned to improve through play, the agent team is not a strong contender in its current state; the best FRIGHT team allowed the Starter Ms. Pac-Man agent to score an average of 4,552 points, which is only slightly better than the starter agents average (5,695 points) against all opponents during the WCCI 2012 Competition [21]. Rather than produce a competitive controller, the goal of this project was to evaluate whether EC could be used to learn coordinated strategies in a distributed multi-agent system, and in this regard it was a success.

VIII. Future Work

The work reported here represents the initial steps of development and evaluation for FRIGHT. We plan on conducting further experiments on the system, such as:

1) We plan to assess and redesign the abstraction of the game used as the set of conditions and actions.

2) We intend to run experiments using varying parameters to the evolutionary strategy, including varying the sizes of the rule sets, the probabilities for the rule set operators, the frequency of don’t care conditions, and the lambda and mu parameters. We plan to run experiments in which we vary the internal FRIGHT parameters (such as the penalties applied to edges) by co-evolving parameter values with rule sets.

3) We also intend to include a variety of Pac-Man agents in the evaluation step, so that the agent does not become overly adapted to the Starter Ms. Pac-Man.

4) Currently, a FRIGHT agent is purely reactive. Adding memory and/or lookahead to the agents may lead to richer behavior.

5) Finally, we would like to explore the use of co-evolution to create heterogeneous agent teams.

ACKNOWLEDGMENTS

This project was made possible by support from NASA and the Maine Space Grant Consortium. We would also like to thank Alan Fitzgerald, Ryan Small, Slawomir Bojarski, and Peter Kemeraitis for the work that inspired FRIGHT. We want to thank Philipp Rohlfshagen, David Robles, and Simon M. Lucas for his helpful suggestions. We also want to thank Phillip Rohlfshagen, David Robles, and Simon M. Lucas for organizing the Ms. Pac-Man vs. Ghosts Competition.

REFERENCES


