Improving Temporal Difference Game Agent Control Using a Dynamic Exploration Rate During Control Learning

Leo Galway, Darryl Charles and Michaela Black

Abstract — This paper investigates the use of a dynamically generated exploration rate when using a reinforcement learning-based game agent controller within a dynamic digital game environment. Temporal Difference learning has been employed for the real-time generation of reactive game agent behaviors within a variation of classic arcade game Pac-Man. Due to the dynamic nature of the game environment initial experiments made use of static, low value for the exploration rate utilized by action selection during learning. However, further experiments were conducted which dynamically generated a value for the exploration rate prior to learning using a genetic algorithm. Results obtained have shown that an improvement in the overall performance of the game agent controller may be achieved when a dynamic exploration rate is used. In particular, if the use of the genetic algorithm is controlled by a measure of the current performance of the game agent, further gains in the overall performance of the game agent may be achieved.

I. INTRODUCTION

Providing a virtual space in which game agents are situated, digital game environments are characteristically nondeterministic, dynamic and typically contain multiple game agents [1, 2]. Subsequently, the Artificial Intelligence (AI) subsystem of a game engine, referred to as Game AI, chiefly facilitates the autonomous selection of behaviors for computer-controlled game agents through game-specific perception, navigation and decision making subsystems [3]. Techniques from both traditional and modern AI have been utilized within game AI for a range of game-dependent problems, including the use of A* for path planning [4], rule-based systems and hierarchies of task-networks for goal planning [5, 6], neural networks for game agent control [7] and the application of evolutionary methods to game AI parameter tuning and testing [8, 9]. Likewise, established programming techniques have also been adopted by game developers and incorporated into game AI, such as the predominant use of finite state machines for game agent control [10]. However, the majority of existing game AI implementations are constrained to the production of predefined decision making and control systems, often leading to predictable and static game agent responses. While such restriction ensures a consistent set of game agent behaviors, there is no capacity for the game agent to learn from experience in order to develop reactive responses to dynamic situations that may arise in real-time during gameplay [3, 4].

In order to generate reactive game agent behaviors the use of Machine Learning (ML) techniques is required [3]. Although a large variety of techniques exist within the ML domain, reinforcement learning presents an approach to agent-based learning that focuses on the interaction between an agent and its environment [11]. Due to the implicit correlation with a game agent learning based on interaction within the game environment, reinforcement learning provides a methodology appropriate for use within digital games. Correspondingly, a degree of success with the application of reinforcement learning for game agent control has been demonstrated within the digital games research literature [12-19].

This paper builds upon the authors’ previous work, in which reinforcement learning was employed for the real-time generation of reactive game agent behaviors within a variation of the game Pac-Man [15, 16]. It aims to improve the performance of the game agent controller within the digital game environment test-bed by dynamically generating a value for the exploration rate of the control algorithm during gameplay using a Genetic Algorithm (GA). By encouraging exploratory action selection during learning, it is expected that action selection will improve during gameplay, thereby enhancing the overall performance of the game agent controller. Details of the experiments will be presented, along with analysis of the results obtained from the corresponding game agent controllers. Subsequently, the results shall be discussed in terms of the game-specific objectives of the game agent.

II. BACKGROUND

A. Pac-Man: A Digital Game Environment Test-Bed

Pac-Man is a classic arcade game in which the primary objective for the player is to achieve as high a score as possible by maneuvering the game agent (Pac-Man) around a 2D maze (level) in order to consume dots, while at the same time avoiding being eaten by four opponent agents (ghosts). During gameplay, if a ghost encounters Pac-Man, a single life is lost and the game ends when all of Pac-Man’s lives are gone. The level also contains a number of energizers, which permit Pac-Man to eat the ghosts for a predefined period of time, thus increasing the score obtained. Upon clearing a level of all dots and energizers, a new level appears in which all dots and energizers are replenished and both Pac-Man and the four ghosts are repositioned in their respective starting locations. The game
also contains score bonuses in the form of fruit that appear in a fixed location for a period of time at random intervals throughout the game, coupled with an extra life for the player if a score of 10000 is achieved. As game-play progresses, the topography of the game environment changes due to the on-going pursuit of consumable by the player, resulting in a reduction in the number of dots and energizers available. Combined with the non-deterministic behavior of the ghosts, the ability of Pac-Man to change the game-play state of the ghosts for a period of time after eating an energizer, and the periodic appearance of bonus fruit, the game environment may be considered as non-stationary and dynamic [20, 21]. From the point of view of the player, the objective of maximizing the overall score obtained within such an environment may be considered as the problem of developing game-play strategies through a combination of navigation, task prioritization and risk assessment [20, 22, 23].

As a suitable test-bed for ML research, the dynamic environment used by such a predator-prey style game presents significant challenges for the automated generation of intelligent game playing agents [20]. In particular, the grid-based environment presented by Pac-Man also offers the advantage of being easily decomposed into a set of states, with a finite state and action space [24]. Subsequently, a number of attempts have been made to incorporate various learning techniques into game AI within variations of Pac-Man in order to implement controllers for both the game agent and opponent agents [15, 16, 20-25]. These include the generation of a game agent move evaluation mechanism for game agent control [25], the utilization of windows of environment features within a local neighborhood of the game agent as input vectors to an evolved multi-layer network game agent controller [20], the use of an evolved probabilistic rule-base coupled with a finite state machine for game agent control [23], evolved neural networks for controlling the opponent agents [21], the use of genetic programming to evolve a series of primitive movement operators for game agent control [22] and the use of reinforcement learning for the real-time generation of a game agent controller [15, 16].

B. Temporal Difference Learning Methods

Based on the discovery of an optimal policy for an agent performing actions within the discrete time, mathematical model of a Markov Decision Process (MDP), reinforcement learning comprises a set of algorithms and techniques that involve learning a sequence of actions in order to maximize an accumulated discounted reward over a period of time, where each action is associated with a reward or penalty. Through exploration and exploitation of an environment by an agent, a control policy may be learned that maximizes the environmental feedback for a sequence of actions without requiring explicit training from a domain expert. Correspondingly, an optimal control policy may be determined by the discovery of an optimal action-value function, which provides an estimation of the value of a given state-action pair by giving the expected long-term return to be gained if the agent begins in the state, takes the action and follows the optimal policy thereafter. Commonly, the action-value function is represented using a look-up table or function approximation method [11-14]. Although a MDP assumes that the environment is stationary, deterministic and contains only one agent, a class of algorithms, known as Temporal Difference (TD) methods, which includes the Q-Learning and Sarsa algorithms, do not require a model of the dynamics of the environment and can be used to overcome the limitations of deterministic MDP environments [11]. An extension of the Sarsa algorithm, Sarsa(λ), incorporates the use of eligibility traces in order to maintain an accumulating trace of state-action pairs visited, which signify the amount of reward a state-action pair is eligible to receive. As each step of learning proceeds, the eligibility trace values for each state-action pair decay in relation to the trace-decay parameter, λ, unless the agent has visited the state-action pair. Although the use of eligibility traces gives rise to larger computation times, they offer the advantages of faster convergence to an optimal policy and permit the use of temporal difference methods when the learning task is partially non-Markov. For a comprehensive discussion on reinforcement learning, please refer to [11].

C. Reinforcement Learning in Digital Games Research

Within the academic digital games research literature, reinforcement learning techniques have been employed to learn control policies in a wide range of game environment test-beds. In [13], Q-Learning was used to determine an optimal policy for game agent control within an N-Person Iterated Prisoners Dilemma test-bed entitled Escape. Making use of a database of game agent states recorded during each simulation step, updates to the control policy were performed every time a game agent interacted with an opponent agent, leading to an improvement in the cooperation between the agents during game-play.

Sarsa(λ) has been utilized in order to generate a near-optimal policy for game agent control within the fighting game Tao Feng: Fist of the Lotus. Investigating both linear and function approximation methods for action-value function representation, it was discovered that an increase in exploration during learning resulted in aggressive action selection by the game agent during game-play, with a corresponding increase in the time taken to learn a control policy [12].

In [26], the Sarsa algorithm was employed for game agent control within a remote control car racing simulation. However, by updating the value function after each simulation step, a suitable control policy was unable to be obtained when either a linear or function approximation value function representation was used.

Sarsa(λ) has also been employed in [15, 16] for the real-time generation of a policy for game agent control within a variation of Pac-Man. Employing a location-based state space, together with environmental feedback corresponding to the high-level features of the game environment, and a linear value function representation, learning was performed prior to action selection during game-play. Subsequently, the generation of reactive game agent behaviors was observed.
Sarsa(\(\lambda\)) has also been used within the strategy game *Settlers of Catan* in order to generate a control policy for game agent strategies. By pre-training the value function with domain specific knowledge, improvements were found in the time taken to learn the control policy, with a subsequent improvement being observed in the game agent behaviors generated [14].

In [17], Sarsa(\(\lambda\)) was also used to generate a control policy for game agent strategies within the real-time strategy game *Battleground*. Employing a hierarchical approach to strategic decision making, the control algorithm learned high-level strategies that were then passed to rule-based game agents in order to perform low-level behaviors. However, it was observed that the use of a linear value function representation limited the strategies that were generated during learning against rule-based opponent agents. Continuing the research in [18], using a function approximation method for value function representation, an overall improvement was observed in the resulting performance of the game agent controller.

A hierarchical approach to reinforcement learning was also used for the generation of policies for game agent control within the real-time strategy game *Battle of Survival*. Decomposing the learning task into a hierarchy of subtasks, standard and modified versions of Q-Learning were employed, with faster convergence to a sub-optimal policy and a corresponding improvement in the resulting game agent performance being observed when the modified algorithm was utilized [19].

D. Evolutionary Optimization in Digital Games Research

A variety of evolutionary methods have been used for the optimization of game agent controller representations within the academic digital games research literature. In particular, the GA has primarily been utilized for both the hybridization and optimization of a range of ML approaches, including rule-bases [27], neural networks [21] and case based reasoning [28].

III. METHODOLOGY

For the experiments discussed within this paper, the TD control algorithm Sarsa(\(\lambda\)) was employed as a game agent controller within a digital game environment test-bed. In order to test the hypothesis that an improvement in the overall performance of the game agent controller may be achieved when the degree of exploratory action selection during learning is dynamically adjusted over the course of a game, three sets of experiments have been conducted, denoted Sarsa(\(\lambda\)), Sarsa(\(\lambda\))-GA and Sarsa(\(\lambda\))-AGA.

In Sarsa(\(\lambda\)), a static value of 0.1 was specified as the exploration rate of the control algorithm for all games played, in order to help maintain a high degree of exploitation during learning. By contrast, in both Sarsa(\(\lambda\))-GA and Sarsa(\(\lambda\))-AGA a canonical GA was utilized to dynamically generate a value for the exploration rate. For Sarsa(\(\lambda\))-GA a new value for the exploration rate was generated prior to each period of learning that was performed during game-play, whereas the use of the GA was adaptively controlled in Sarsa(\(\lambda\))-AGA. During each set of experiments, a series of 20 independent games were played for each setting specified over the range [0, 1] for the trace-decay parameter of the control algorithm. A look-up table was used for the action-value function representation, which was arbitrarily initialized with values in the range [-0.1, 0.1] prior to each game played. Further details of the experimental setup shall now be discussed.

A. Dynamic Digital Game Test-bed Implementation

For all experiments conducted, the test-bed was a variation of the classic arcade game *Pac-Man* implemented as a 20x20 grid, where each grid cell contains a single high-level feature of the game environment, such that the configuration of features establishes the topography of the level used. Additionally, from the point of view of the game agent, the opponent agents may also be considered as one of the high-level features of the game environment. Subsequently, the complete set of features comprises; walls, dots, energizers, invalid spaces (i.e. grid cells for the starting location of the opponent agents), empty spaces (i.e. grid cells where a dot or an energizer was consumed) and opponent agents.

Both the game agent and 4 opponent agents begin each game in predefined starting locations. During game-play, if the game agent consumes one of the energizers, the gameplay state associated with each of the opponent agents temporarily changes from a default *Attack* state to an *Evade* state, during which the game agent may eat the opponent agents. The duration of the Evade state has been predefined as 300 simulation steps, which may be further reset to a maximum possible duration of 300 simulation steps every time one of the remaining energizers is consumed. Any opponent agent consumed by the game agent is regenerated at its default starting location during the next simulation step and the game-play state of the opponent agent is subsequently reset to the default state. Conversely, if an opponent agent encounters the game agent, when the gameplay state is the *Attack* state, the game agent loses a life and is regenerated at its default starting location in the next simulation step.

For both the game agent and the opponent agents, moves may be made in the North, South, East and West directions, with the game agent making a move 50% more often than the opponent agents. In the experiments, the choice of moves for the opponent agents were non-deterministic, with sets of moves for each opponent agent being randomly pre-generated, thus preventing the learning algorithm from simply learning a fixed set of movement patterns. In order to allow for a direct comparison between each series of games played, the same set of opponent agent moves were used for each series of games and the game agent was restricted to a maximum of 1000 moves per game.

Unlike the arcade version of the game, a single level has been used for all games played, which was initially populated with a total of 176 dots and 4 energizers. Additionally, no bonuses were awarded, either in terms of bonus fruit or extra lives, within the test-bed implementation. A game ends when all the game agent’s
lives are gone, the total set of dots, including energizers, has been consumed by the game agent, or the maximum number of moves for the game agent has been reached.

B. Sarsa(\(\lambda\)) Game Agent Controller Implementation

During game-play, the choice of moves performed by the game agent was based on the outcome of real-time learning using the Sarsa(\(\lambda\)) control algorithm. As the game environment was designed as a 20x20 grid, a location-based state space abstraction was employed, where each grid cell was considered as a state containing 4 possible actions, with each action representing one of the moves that may be made by the game agent. Consequently, the size of the state-action space was 1600. Values representing the complete set of high-level game environment features, including separate values for an opponent agent in either the Attack or Evade game-play states, were utilized as environmental feedback by the reward function of the control algorithm. Throughout the course of a game, every time a move was required for the game agent a period of learning (control learning) was first conducted, in which the algorithm was run for 100 episodes of learning using a copy of the current topography of the game environment before a move was chosen for actual game-play. During control learning, actions selected by the algorithm could potentially lead to the consumption of dots and energizers within the copy of the game environment however, the location and state of the opponent agents remained fixed for the duration of control learning. The initial state for each episode of control learning was specified as the world-space coordinates of the grid cell corresponding to the current location of the game agent. During each episode an action was selected using an \(\epsilon\)-greedy policy, with the immediate reward received specified according to the feature encountered in the grid cell corresponding to the new state observed after the action was performed. The action-value function and eligibility trace values were then updated. Subsequently, after 100 episodes of learning were performed a game-play move was chosen for the game agent by selecting the state-action pair, for the state corresponding to the current location of the game agent, using a greedy policy with respect to the updated action-value function.

C. Genetic Algorithm Implementation

For both Sarsa(\(\lambda\))-GA and Sarsa(\(\lambda\))-AGA, a canonical GA was incorporated into the game agent controller in order to generate a value for the exploration rate parameter of the control algorithm prior to each period of control learning. As such, the GA comprised an 8-bit binary chromosome representation scheme, roulette wheel selection and a unary crossover operator. Each chromosome represented the fractional component of a value in the range \([0.10, 0.90]\) with an 8-bit binary fraction. Due to the fixed range of possible values for the exploration rate, each chromosome was subsequently limited to a set of binary strings in the range \([00011001, 11100111]\), thus representing the range \([0.09765265, 0.90234735]\). Consequently, the binary range was enforced by the genotype-to-phenotype decoding during fitness evaluation, with any phenotype values outside the range \([0.09765265, 0.90234735]\) being assigned either the respective maximum or minimum value, and the corresponding genotype written back into the associated chromosome before fitness evaluation was performed. The fitness function employed by the GA mimicked the control algorithm over a single episode of control learning within a static copy of the current topography of the game environment. During fitness evaluation, all features of the game environment remained static, including the locations of the opponent agents. Both the current game-play location of the game agent and the static copy of the game environment were utilized by the fitness function throughout all generations of the GA. Neither the action-value function, nor the eligibility values of the control algorithm were updated during fitness evaluation, as illustrated in Fig. 1.

![Fitness Evaluation Algorithm](image)

As may be observed in Fig. 1, the genotype was first decoded into its corresponding phenotype and used as the exploration rate by the fitness function. An action was then selected for the state corresponding to the current location of the game agent, using the control algorithm’s action selection mechanism and current policy. The immediate reward value for the subsequent state transition was obtained and assigned as a fitness value for the chromosome.

During each generation of the GA, all chromosomes within the current population were evaluated and assigned fitness values. Linear scaling was then performed on the resulting set of fitness values and a number of chromosomes were selected for inclusion in a mating pool using roulette wheel selection. In order to generate a new population of chromosomes, two parent chromosomes were chosen randomly from the mating pool and either copied directly into the new population or combined, using single-point crossover, to produce an offspring which was then added to the new population. Mutation was then performed on the individual bits of each chromosome within the new population. The process of parental selection, crossover and mutation were repeated until a new population of chromosomes was generated. After a predefined number of generations, the fittest chromosome obtained over all generations was decoded and used as the exploration rate by the learning algorithm in the next period of control learning.

For all experiments conducted, a relatively small population size (10) and total number of generations (50) were employed in an effort to permit the generation of a value for the exploration rate to be performed in real-time. The GA also utilized a crossover rate of 0.6, together with a mutation rate of 0.02.
D. Adaptive Genetic Algorithm Implementation

In Sarsa(λ)-AGA, a new value for the exploration rate was only generated when the overall performance of the game agent did not increase between consecutive moves during game-play. In order to provide adaptation, the performance of the game agent, \( \text{Performance}_i \), was determined as:

\[
\text{Performance}_i = \frac{S_i + D_i + E_i + (OD_i - TK_i)}{P}
\]

where, \( S_i \) is the current score, \( D_i \) is the total number of dots and energizers consumed, \( E_i \) is a measure of the current entropy of the game agent, \( OD_i \) is the total number of opponent agents defeated, \( TK_i \) is the total number of game agent fatalities and \( P \) is the total number of moves. The entropy metric, \( E_i \), was utilized to indicate the current spatial diversity of the game agent over the game environment, defined as [21]:

\[
E_i = -\sum_1^P \frac{P_i}{P} \log_2 \left( \frac{P_i}{P} \right)
\]

where \( p_i \) is a count of the number of times a specific grid cell is visited by the game agent and \( P \) is the overall total number of moves that have been performed by the game agent.

Prior to each period of control learning, before the game agent selects a move during game-play, the current performance was calculated and compared to the performance value obtained before the previous move. If a decrease in the performance occurred, such that \( \text{Performance}_{i-1} \geq \text{Performance}_i \), the GA was used to determine a new exploration rate, otherwise the control algorithm utilized the current exploration rate during the ensuing period of control learning. The GA has also been used to determine the initial value for the exploration rate as no performance value may be obtained before the first move of each game.

IV. RESULTS

As a single level has been used in the experiments conducted, the game agent may consume a maximum of 180 dots over the course of a single game: 176 dots plus 4 energizers. Similarly, for each energizer consumed by the game agent, a maximum of four opponent agents may be eaten. Subsequently, the maximum possible score obtainable for a single game is 2680 points. In the presentation and analysis of the results from each set of experiments, the results for the score and the number of dots consumed shall be discussed in terms of the percentage of the maximum possible score obtained (%Score) and the percentage of the maximum possible number of dots consumed (%Dots) over a single game.

Fig. 2 illustrates a comparison of the results obtained from Sarsa(λ), Sarsa(λ)-GA and Sarsa(λ)-AGA for the mean %Score, over each series of 20 games played. In Fig. 2 it may be observed that the results are larger, in the majority of series of games played, when the exploration rate was dynamically generated. Consequently, over each set of results obtained from Sarsa(λ), Sarsa(λ)-GA and Sarsa(λ)-AGA, the values for the average %Score were 37.46, 42.51 and 43.25 respectively. The corresponding standard deviation values were determined as 12.52, 9.58 and 8.88, indicating that the most consistent set of results was also obtained when the use of the GA was adapted to the current performance of the game agent controller.

When the set of results obtained from Sarsa(λ)-GA was compared with the results obtained from Sarsa(λ), an average increase in the mean %Score of approximately 39.01% was determined over each series of games played in which an increase in the result was observed. Likewise, when the set of results obtained from Sarsa(λ)-AGA was compared with the results from Sarsa(λ), an average increase of approximately 41.13% was found. In conjunction with the results illustrated in Fig. 2, this would further indicate that the adaptive use of the GA improves the results obtained for the mean %Score over the use of a static value for the exploration rate, regardless of the trace-decay parameter utilized by the control algorithm.

A comparison of the results obtained from each set of experiments for the mean %Dots consumed is subsequently given in Table I.

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>Sarsa(λ)</th>
<th>Sarsa(λ)-GA</th>
<th>Sarsa(λ)-AGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>95.86</td>
<td>96.06</td>
<td>97.83</td>
</tr>
<tr>
<td>0.1</td>
<td>96.06</td>
<td>97.47</td>
<td>96.44</td>
</tr>
<tr>
<td>0.2</td>
<td>96.47</td>
<td>95.62</td>
<td>93.56</td>
</tr>
<tr>
<td>0.3</td>
<td>95.66</td>
<td>95.64</td>
<td>96.61</td>
</tr>
<tr>
<td>0.4</td>
<td>95.97</td>
<td>97.17</td>
<td>95.17</td>
</tr>
<tr>
<td>0.5</td>
<td>92.42</td>
<td>95.19</td>
<td>91.28</td>
</tr>
<tr>
<td>0.6</td>
<td>91.86</td>
<td>93.53</td>
<td>96.28</td>
</tr>
<tr>
<td>0.7</td>
<td>85.25</td>
<td>93.22</td>
<td>94.11</td>
</tr>
<tr>
<td>0.8</td>
<td>83.61</td>
<td>94.47</td>
<td>92.53</td>
</tr>
<tr>
<td>0.9</td>
<td>59.47</td>
<td>74.31</td>
<td>79.31</td>
</tr>
<tr>
<td>1.0</td>
<td>12.75</td>
<td>39.39</td>
<td>41.86</td>
</tr>
</tbody>
</table>

In contrast to the results for the mean %Score, it may be observed from Table I that no individual experiment, using either Sarsa(λ)-GA or Sarsa(λ)-AGA, was able to produce a set of results for the mean %Dots which improved upon the results obtained from Sarsa(λ) for all settings specified for
the trace-decay parameter. However, the values for the average %Dots were determined as 82.39, 88.43 and 88.63 for Sarsa(λ), Sarsa(λ)-GA and Sarsa(λ)-AGA respectively, thereby indicating that an improvement occurred in the number of dots consumed when a dynamic exploration rate was utilized. The corresponding standard deviation values for each set of results were 25.55, 17.51 and 16.31, signifying that adaptive use of the GA produced the most consistent set of results. Over the series of games played for Sarsa(λ)-GA and the series of games played for Sarsa(λ), in which an improvement in the results was observed, an average increase in the mean %Dots of approximately 29.33% was determined. Similarly, for Sarsa(λ)-AGA, an average increase of approximately 36.26% was observed.

From the analysis of the results for both the mean %Score and the mean %Dots consumed, it is apparent that a greater improvement has occurred in the set of results obtained for the mean %Score, than in the corresponding set of results for the mean %Dots, when the GA has been used to determine the exploration rate. As the score takes both the number of dots consumed and the number of opponent agents defeated into account, this would suggest that the dynamic generation of an exploration rate has a more beneficial effect on the opponent agent pursuit behaviors generated by the game agent controller, than on the ability of the game agent to successfully navigate the game environment. Subsequently, a comparison of the results obtained for the total number of opponent agents defeated over each series of 20 games played is illustrated in Fig 3.

In Fig. 3 it may be observed that the majority of results obtained from Sarsa(λ)-GA and Sarsa(λ)-AGA have improved upon the corresponding results obtained from Sarsa(λ). Consequently, the total number of opponent agents defeated over all games played for Sarsa(λ), Sarsa(λ)-GA and Sarsa(λ)-AGA were 260, 407 and 443 respectively. Comparing the set of results obtained from Sarsa(λ)-GA with the set of results obtained from Sarsa(λ), an average increase of approximately 103.58% was found over the majority of series of games played when the exploration rate was generated using the GA. Likewise, over the entire set of results obtained from Sarsa(λ)-AGA an average increase of approximately 103.30% was determined. Furthermore, when the set of results obtained from Sarsa(λ)-AGA was compared with the set of results from Sarsa(λ)-GA, an average increase in the total number of opponent agents defeated of approximately 26.46% was found over those series of games played in which an improvement in the result was observed. As such, the adaptive generation of a value for the exploration rate of the control algorithm has once again shown the greatest improvement over the results obtained when a static value for the exploration rate has been used during control learning throughout the course of a game.

Fig. 4 presents a comparison of the results obtained from each set of experiments for the total number of times the game agent was killed over each series of 20 games played.

In accordance with the results for the total number of opponent agents defeated, the results from Sarsa(λ)-GA and Sarsa(λ)-AGA show a general improvement over the results obtained when a static value was used for the exploration rate. However, in Fig. 4 it may be observed that no single set of results shows an improvement over the set of results from Sarsa(λ) for all the settings specified for the trace-decay parameter. Correspondingly, the total number of game agent fatalities over all series of games played for Sarsa(λ) was 487, whereas Sarsa(λ)-GA and Sarsa(λ)-AGA attained overall total values of 421 and 364 respectively. Comparing the set of results obtained from Sarsa(λ)-GA with the corresponding results from Sarsa(λ), an average decrease in the total number of times the game agent was killed of approximately 19.98% was determined over the majority of series of games played. Similarly, for Sarsa(λ)-AGA an average decrease of approximately 26.98% was found. Again, when the set of results obtained from Sarsa(λ)-AGA was compared with the set of results obtained from Sarsa(λ)-GA, an average decrease of approximately 17.99% was observed.

Consequently, when a GA has been adaptively utilized for the generation of the exploration rate during control learning, less game agent fatalities occurred during gameplay. Previously it was suggested in the analysis of the results for the mean %Score and the mean %Dots, that the improvement observed in the results for the mean %Score was primarily due to a corresponding improvement in the opponent agent pursuit behaviors generated by the game agent controller. From the results obtained for both the total
number of opponent agents defeated and the total number of times the game agent was killed, as illustrated in Fig. 3 and Fig. 4 respectively, it has been shown that, in general, the ability of the game agent to pursue and avoid the opponent agents has been improved when a dynamic exploration rate was utilized during control learning. Although marginal improvements have also been observed in results obtained from Sarsa(λ)-GA and Sarsa(λ)-AGA for the mean %Dots consumed, thereby partially improving the results for the mean %Score, the substantial enhancement found in the opponent agent pursuit behaviors supports the previous suggestion.

Fig. 5 presents a comparison of the results for the mean time taken per game, in seconds, from Sarsa(λ), Sarsa(λ)-GA and Sarsa(λ)-AGA.

![Fig. 5. Comparison of Mean Time Taken Per Game](image)

In Fig. 5 it may be observed that in the majority of series of games played, the mean time taken per game is reduced when a dynamic exploration rate has been utilized during control learning. Correspondingly, the average time taken per game for Sarsa(λ), Sarsa(λ)-GA and Sarsa(λ)-AGA was 99.35, 95.21 and 95.12 respectively. Although a degree of fluctuation can be seen across each set of results depicted in Fig. 5, it may also be observed that the results obtained from Sarsa(λ)-AGA are lower when \(0.0 \leq \lambda \leq 0.5\), than the corresponding results obtained from Sarsa(λ)-GA. However, for the series of games played when \(0.6 \leq \lambda \leq 1.0\), a marginal increase occurred in the result from each series of games played. Subsequently, Sarsa(λ)-AGA produced results in the range [86.93, 105.22], whereas the range of results obtained from Sarsa(λ)-GA was [90.09, 99.60]. Even though a smaller range of results was achieved when the GA was used prior to each period of control learning, there was a marginal decrease in the average time taken per game when adaptation was utilized. Consequently, when the set of results obtained from Sarsa(λ)-AGA was compared with the set of results from Sarsa(λ)-GA, an average decrease in the mean time taken of approximately 4.81% was determined over the series of games played when \(0.0 \leq \lambda \leq 0.5\). By contrast, over the series of games played when \(0.6 \leq \lambda \leq 1.0\), an average increase of approximately 5.68% was found.

For Sarsa(λ) the set of results were produced in the range [91.67, 105.42]. In comparison to the sets of results obtained when a dynamically generated exploration rate was used, average decreases in the mean time taken of approximately 5.85% and 5.77% were determined for Sarsa(λ)-GA and Sarsa(λ)-AGA respectively. This may be potentially due to a greater number of games played in which the game agent clears the level of dots and energizers, thereby reducing the time taken to complete a game. As the corresponding results for the mean %Dots have illustrated an overall improvement over the results obtained from Sarsa(λ), as previously given in Table I, this would indicate that a larger proportion of games have been played in which the level has been cleared by the game agent.

From the results presented in Fig. 2 and Table I, together with the comparison of results given in Fig. 3, Fig. 4 and Fig. 5, it may be observed that, in general, the adaptive use of a GA for the dynamic generation of an exploration rate has improved the overall performance of the game agent controller in comparison to the use of a static exploration rate during control learning. Furthermore, the overall performance has been shown to be superior to the performance obtained when the GA was used before every period of control learning. As adaptation, based on the difference in the overall performance of the game agent between successive moves during game-play, has been utilized to control the generation of new values for the exploration rate, then if no decline is observed in performance, the exploration rate used during the previous period of control learning will be subsequently used during the current period of learning. Subsequently, when the exploration rate has been optimized with respect to the current topography of the game environment, it is more likely to remain appropriate over consecutive periods of control learning.

V. CONCLUSION

It was hypothesized that the use of dynamic exploration rate, over the course of a game, would improve the action selection performed by the control algorithm as game-play progressed, thereby enhancing the overall performance of the game agent controller. By dynamically adjusting the amount of exploratory action selection during control learning, as the number of game environment features which provide positive environmental feedback diminishes throughout the course of a game, beneficial state-action pairs may be discovered that lead to the improved selection of actions by the game agent during game-play. Experiments conducted utilizing a GA for the optimization of the exploration rate, prior to each period of learning, improved the overall performance of the game agent controller, with adaptive control of the GA yielding further performance gains.

Investigations were conducted in which a GA was incorporated into the game agent controller with the aim of optimizing the exploration rate with respect to the reward value received for a state transition based on the action selection mechanism of the control algorithm. Subsequently, it was demonstrated that the real-time generation of a value for the exploration rate was successful in improving the
overall performance of the game agent controller. However, in these experiments, the GA was used prior to every period of control learning during game-play. As this may lead to a decrease in the computational efficiency of the control algorithm, a further set of experiments were conducted which aimed to control the use of the GA by integrating a degree of adaptation into the game agent controller. Using a predefined measure of the current performance of the game agent controller, the control algorithm was modified to only generate a value for the exploration rate when the overall performance of the game agent dropped between successive moves during game-play. Results demonstrated that the adaptive use of the GA further improved the overall performance of the game agent controller. Moreover, the performance improvement observed was superior to the corresponding improvement obtained when the GA was employed prior to every period of control learning.

In conclusion, due to the dynamic nature of the digital game environment test-bed, a dynamic exploration is essential for the enhanced performance of the game agent controller. Such a dynamic exploration rate may be readily obtained in real-time using a canonical GA in collaboration with the control algorithm. A potential limitation of the experiments presented is the use of a single objective fitness measure for the GA. By utilizing a multi-objective fitness function, a larger range of fitness values may be potentially obtained for each population of chromosomes, thereby improving the convergence of the GA. Investigations should also be conducted into the parameters of the GA in an effort to further improve convergence. The research presented could also be extended to the real-time optimization of the discount rate utilized by the control algorithm, thereby permitting the control algorithm to automatically adjust the emphasis placed on rewards received according to the current performance of the game agent.

REFERENCES


