Dynamic Game Difficulty Scaling Using Adaptive Behavior-Based AI

Chin Hiong Tan, Kay Chen Tan, and Arthur Tay

***Abstract*—Games are played by a wide variety of audiences. Different individuals will play with different gaming styles and employ different strategic approaches. This often involves interacting with nonplayer characters that are controlled by the game AI. From a developer's standpoint, it is important to design a game AI that is able to satisfy the variety of players that will interact with the game. Thus, an adaptive game AI that can scale the difficulty of the game according to the proficiency of the player has greater potential to customize a personalized and entertaining game experience compared to a static game AI. In particular, dynamic game difficulty scaling refers to the use of an adaptive game AI that performs game adaptations in real time during the game session. This paper presents two adaptive algorithms that use ideas from reinforcement learning and evolutionary computation to improve player satisfaction by scaling the difficulty of the game AI while the game is being played. The effects of varying the learning and mutation rates are examined and a general rule of thumb for the parameters is proposed. The proposed algorithms are demonstrated to be capable of matching its opponents in terms of mean scores and winning percentages. Both algorithms are able to generalize well to a variety of opponents.**

***Index Terms*—Artificial intelligence, behavior based, car racing simulation, game AI, player satisfaction, real-time adaptation.**

I. Introduction

GAMING is by definition an interactive experience [1]. It involves interacting with both the game environment and nonplayer characters (NPCs) that are controlled by the game artificial intelligence (AI). In this paper, we will examine the interaction between the player and the game AI.

High-quality game AI has become an important selling point of computer games in recent years [2]. However, game players still prefer to play against human controlled opponents (via a network) rather than computer controlled ones. Indeed, multiplayer support and playing against human opponents over the Internet has become the norm. This is because the gaming community feels that the quality of game AI is still generally low [3]. For instance, game AI is far from human abilities in complex games such as *Go* [4]. Nevertheless, there exist some situations where an entertaining game AI with high replay value is still desirable. An example of such a situation is in the absence of a viable network connection (e.g., public buses, commercial flights, cruises), or simply when the game player is in the mood for a quiet game alone.

A given game is played by a wide variety of audiences who play with different gaming styles and use different strategic approaches. Thus, a static game AI is unlikely to be able to cater to the playing styles of all types of players. An adaptive game AI, on the other hand, has the potential to create a different game experience for different players, and thereby add value and replayability to a game. A study with human players conducted by Hagelback and Johansson also demonstrated that players found it more enjoyable to play an even game against an opponent that adapts to the performance of the player [29]. Hence, the objective of this paper is to develop an adaptive game AI that tries to entertain its opponent by playing an even game rather than to defeat him.

Adaptive game AI refers to a dynamic computer controlled player that adapts its game behavior in response to its opponents, either during the game playing session or in between sessions. In particular, dynamic game difficulty scaling uses adaptive game AI to automatically adapt game parameters and behaviors in real time according to the proficiency of the player in the game. It has the potential to keep the player interested for a longer period of time and improves the playing experience. Adaptive mechanisms in games have been actively explored in recent years. Togelius *et al.* used evolutionary algorithms to evolve racing tracks that maximized the entertainment value to particular human players [5], [6]. Spronck *et al.* introduced an adaptive algorithm that used an adaptive rulebase that can be used with current scripting game AI [7], [23]. Bergsma and Spronck implemented allocation and decomposition architecture for performing tactical AI (ADAPTA) that can learn and defeat static opponents in combat for a turn-based strategy game [8]. Bryant and Miikkulainen used neuroevolution to evolve a team of adaptive agents that can learn and adopt strategies in a strategy game [10]. Stanley *et al.* used a real-time version of NeuroEvolution of Augmenting Topologies (rtNEAT) to allow agents in a game to adapt and improve during the game [11]. Yannakakis used evolutionary machine learning to exploit cooperative behaviors that can increase a player's interest while playing [12]. Yannakakis and Hallam also implemented an adaptive *Bug-Smasher* game that improved the satisfaction of children that played it [28]. Thue *et al.* used an interactive storytelling system that models a player automatically to dynamically select content to create an interactive story [9]. Riedl and Stern developed an automated story director that can adapt the plot of a story even when the player lands in an unexpected scenario [26]. Barber and Kudenko proposed an adaptive narrative engine that is able

Manuscript received August 28, 2010; revised January 04, 2011 and April 27, 2011; accepted May 19, 2011. Date of publication June 02, 2011; date of current version December 14, 2011.

C. H. Tan is with the Institute for Infocomm Research, Agency for Science Technology and Research (A*STAR), Singapore 138632, Singapore (e-mail: chtan@i2r.a-star.edu.sg).

K. C. Tan and A. Tay are with the Department of Electrical & Computer Engineering, National University of Singapore, Singapore 117576, Singapore (e-mail: eletankc@nus.edu.sg; eletaya@nus.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2011.2158434

to automatically generate story events based on the interactions and decisions made by the user [34]. Balaji and Srinivasan developed an adaptive intelligent signal control framework using multiagent systems for a simulated traffic network [35]. Quek *et al.* used coevolutionary learning as a means of adaptation to study agent interactions in a public goods game that can be used in the genre of business simulation games [27]. Tan *et al.* experimented with adaptive rules for a minimax search tree to adapt to its opponents in *Gomoku* [22]. Kim *et al.* used incremental multiobjective learning to find the optimal path to the ball in a robot soccer platform [23]. Durr *et al.* demonstrated how an evolutionary algorithm can be used to automatically adapt modular neural networks topologies in a maze learning task [24], [37]. Andrade *et al.* used a reinforcement learning approach to quickly identify and track the proficiency of a human player in a real-time fighting game [25]. Wang *et al.* used a reinforcement learning algorithm to improve a team of bots against its opponents in *Unreal Tournament* [33].

This paper focuses on the adaptation of the game AI during a game session. In other words, the difficulty scaling is done in real time. The adaptive game AI needs to be smart enough to make unpredictable but rational decisions like human players do, but should not display obviously stupid behavior such as being stuck in an endless loop. The adaptive game AI should also be able to profile its opponent efficiently during the early phase of the game and adapts its own playing style to the proficiency of the player so that the player feels entertained playing against the AI. This paper presents two adaptive algorithms that use ideas from reinforcement learning [31], [32] and evolutionary computation to play adaptively in a real-time driving game to provide the opponent with a competitive and entertaining experience. Two indicators, namely, mean score difference and winning percentage difference, are proposed as a measure of entertainment value.

This paper is organized into the following sections. Section II provides a brief description of the simulated car racing environment and the behavior-based system that forms the basis of the adaptive controller. Section III presents the indicators used to measure entertainment value and also the mechanisms of the adaptive algorithms and its implementations. Section IV introduces the different static opponents that will be used as test subjects for the adaptive controllers. Section V describes the experiments conducted and discusses their results. Conclusions are drawn in Section VI.

II. Simulated Car Racing

This section describes the car racing simulator that is used as the problem domain for this paper. It also describes and introduces additional behaviors to the basis controller used to develop the online adaptive controller.

A. Simulator Model

The car racing simulator model [13] used in this experiment is modified from the Simulated Car Racing Competition held during the 2007 IEEE Congress on Evolutionary Computation (CEC 2007) [14]. The main features of the simulator will be summarized in this section.

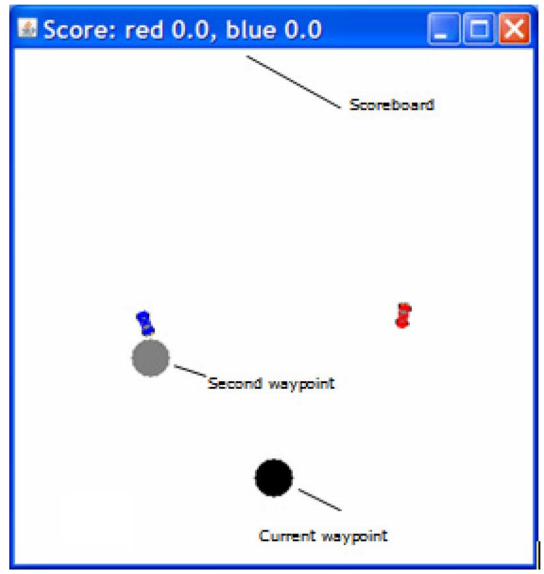

Fig. 1. Playing area for simulated car racing. The red and blue cars are the first and second players, respectively. The current waypoint, which is in black, is worth one point if it is passed by either car. The second waypoint, which is in gray, is worth zero points at this moment. However, once the current waypoint is passed, the second waypoint will be promoted to be the current waypoint and it is then worth one point. A new second waypoint will then be generated.

The dimension of the competition field is 400 pixels by 400 pixels and is not occupied by any walls or obstacles as shown in Fig. 1. As such, vehicles are free to drive outside the competition field. However, only the competition field is visible to the human controlled player. Hence, this setup is advantageous to computer controlled cars as their sensors will continue to function even if the current position of their car is outside the field.

The objective of each race is to pass as many waypoints as possible within an allotted time. Two waypoints are visible on the competition field: the current and the next. However, the next waypoint cannot be collected unless the current is passed. The method of generating waypoints in this model differs from that of the original which was generated randomly around the field. The main disadvantage of randomly generated waypoints is that two or more sequential waypoints may be generated in close proximity of one another and in severe cases, even overlapping. This results in a particular vehicle gaining two or more points in a single approach. This can be viewed as a biased allocation of points especially in a race where collecting waypoints is its main objective. As such, the following method of generating waypoints is proposed. The locus of the $(k+1)$th waypoint always lies on the circumference of a circle of radius (400/3) pixels centered on the kth waypoint within the visible competition area. The arbitrarily chosen radius is to ensure waypoints do not overlap while maintaining some distance between waypoints to allow opposing vehicles an opportunity to mount a viable counter strategy. In addition, the very first waypoint is always initialized on the locus of a vertical straight line running through the center of the field. This is to ensure that the initial condition is not in favor of any vehicle in particular.

The vehicles themselves are controlled using discrete control actions that can be mapped to the directional controls found on

console game control pads or the cursor arrow keys on the keyboard. The four distinct on/off control signals, accelerate, decelerate, left and right turn, combine to form a total of nine possible control actions, including a neutral action. The controller can take inputs, either from the keyboard or an AI algorithm. The included physics model is reasonably detailed, allowing for collisions between vehicles as well as side skidding. A technically skilled controller, human or not, will be able to execute such maneuvers when cornering to their advantage.

B. Behavior-Based Controller

The behavior-based controller [15] will be used as the basis controller to develop the adaptive controller in this paper. The various mechanisms of this controller will be briefly discussed in this section as they form the basis for the implementation of the proposed adaptive controller.

The behavior-based controller [15] is inspired by the behavior-based AI [16] commonly used in the field of robotics. It is made up of four independent driving behavior components that are aimed at improving the driving performance of the controller and one tactical behavior component that seek to outplay the opponent in the game. Each behavior component can be activated or deactivated to vary the overall behavior of the controller.

Two additional tactical behavior components are introduced in this paper to take advantage of the dynamism of such a two-player competitive game. Driving behavior components ignore the existence of the opponent in the playing field, leading to inferior performance. Conversely, tactical behavior components help the controller plan and decide which waypoint to head toward or even whether to go for any waypoint at all. As such, driving behavior components can often be viewed as lower level operational intelligence while tactical behavior components can be viewed as higher level decision making intelligence [17]. More details about the driving and tactical behavior components can be found in [30].

An advantage of the behavior-based system used in this controller is its scalability. New behavior components can easily be added or removed from the existing set of behavior components, be they complementary or conflicting. The adaptive algorithm will automatically select a combination of behavior components suitable for its opponent.

Visually, the behavior-based controller can be summarized as follows. First, it decides whether to drive toward the current waypoint or the next waypoint depending on game situations. Next, it will decide whether to drive forward or backward toward its intended destination. Then, it accelerates and drives toward its destination. It may also decide to switch destination at any point in time.

Each behavior component will be described in more detail so as to facilitate the discussion in the later sections.

Driving behavior components are as follows.

1) Hyperbolic Tangent Speed Regulator: The speed of the car is regulated by a hyperbolic tangent function of the distance away from its destination. It provides cues to accelerate, decelerate, cruise at constant velocity, and stop, depending on its distance from the destination. For distances more than 113 pixels away, the car tries to maintain a cruising speed of 7 pixels per time step. For distances smaller than that, the car tries to maintain a lower speed before coming to a stop within 15 pixels to its destination.

2) Reversing: The angle to the destination is used to determine whether to drive forward or reverse in a given situation. If the angle is within a given threshold of -2.5513 to 2.5513 rad, centered in front of the car, the speed regulator will be used as described, otherwise it will be negated and the controller will reverse the car toward the destination instead.

3) Direction Switching Compensation: For instance, if the car is moving backwards and the destination is in the forward right heading, then steering right at this point will instead orientate the car farther to the left, increasing the difference in heading. Instead, the controller should steer left until the reversing motion comes to a halt before steering right and applying the accelerator.

4) Tight Angle Turning: Occasionally, when the turning angle is too small, the controller gets stuck in an orbit around the destination point and stays in orbit. This is partly due to the nature of the on/off controls used in the simulator. To overcome this problem, a manual pulse width modulation technique is used to lower the acceleration during tight turns to avoid being trapped.

Tactical behavior components are as follows.

5) Waypoint Prediction: This is a predictive module that chooses which waypoint is more advantageous for the controller to head toward. By observing the state of the game, this behavior predicts which car will reach the current waypoint first. In the event that the opponent is predicted to be faster to the current waypoint, the controller should then direct the car toward the next waypoint instead and *vice versa*.

6) Time Wasting: This is the first of two newly added behavior components. In time constrained games such as soccer, the team in possession of the ball may choose to pass the ball around and not commence any attacks. This strategy is usually used when the team in possession of the ball is in the lead and wishes to defend their lead. In simulated car racing, the controller may choose to stop at the edge of the current waypoint and not collect it if the opponent is sufficiently far away or is heading toward the next waypoint. This forces the opponent to approach the current waypoint and lose the advantage of heading toward the next waypoint.

7) Blocking: This is the second of two newly added behavior components. When both cars are headed toward the current waypoint, the controller may choose to drive on the path between the opposing car and the waypoint, hence blocking it from the opponent. In the event of a collision, the controller receives a velocity boost toward the current waypoint, hence increasing its chances of reaching the waypoint before its opponent does. Furthermore, if the controller also activates the reversing behavior component, it may be able to recover from a collision faster than the opponent.

III. Adaptive Controllers

This section describes in detail the evaluation criteria used to evaluate the performance of the adaptive controllers. Two adaptive controller algorithms, the adaptive uni-chromosome

controller (AUC) and the adaptive duo-chromosome controller (ADC), will also be introduced and discussed.

A. Satisfying Gameplay Experience

A game experience is considered satisfying or entertaining when it is difficult to defeat [18]. This may be applicable to advanced players but may not necessarily apply for beginners or casual gamers. For casual gamers, the game is most entertaining when it is challenging yet beatable [19]. Malone also pointed to challenge as one of the categories that makes games fun [20]. That is, the game should neither be too easy nor too difficulty. A study with human players conducted by Hagelback and Johansson also demonstrated that players found it more enjoyable to play an even game against an opponent that adapts to the performance of the player [29]. In other words, in a two-player competitive game, the player and his opponent should be evenly matched and the win/loss margin in each game should be small. Spronck *et al.* used a top culling technique to train a game AI to play an even game with its opponent [23]. However, their method required a training period of 50 encounters. Our proposed adaptive algorithms in this paper have the advantage of not requiring a training phase as the adaptation is achieved during the game session.

Csikszentmihályi's theory of flow proposed that how much an opponent is perceived to be challenging depends on the skill of the player in playing the game [35]. In addition, in order to maintain the state of flow, the player will need to seek or be presented with greater challenge. An adaptive opponent that grows together with the player will be able to fulfill this role by constantly adapting to the player. Since the adaptation is performed during each game session, the algorithm will always try to match the current proficiency of the player. This way, the challenge can be maintained.

In the context of a simulated car racing game, there can be three possible outcomes: win (W), lose (L), or draw (D). For this paper, the performance measure of an even game described by Spronck *et al.* is used. That is, a game where the challenge matches the proficiency of the human player is experienced as more entertaining than a game that is either too easy or too hard [23]. Therefore, in a set of n games, the player is considered most satisfied when $W = L = (n - D)/2$ where W is the number of player wins, L is the number of player losses, and D is the number of drawn games. In this paper, two indicators are introduced to measure the satisfaction a player derives from the game.

1) $|W - L|$ should be minimized and D should also be minimized. That is, an even game is one when the player wins and loses an equal number of times over a series of n games. A high number of draws is deemed as more frustrating than fun.
2) $|s1 - s2|$ should be minimized and $\max(s1, s2)$ should be maximized, where $s1$ and $s2$ are the average individual scores of player 1 and player 2 over n games, respectively. A small difference between $s1$ and $s2$ indicates a similar proficiency of play. In addition, small differences imply that both players are engaged in the game and therefore improve satisfaction. Next, a high average score indicates a competitive and fast paced game which also helps to keep players engaged. Although low average scores can also produce competitive games, an informal survey with a few play testers revealed that in the context of a simulated car racing game, high average scores are more entertaining.

1	2	3	4	5	6	7
0.8	0.6	0.1	0.3	0.9	0.5	0.2

Fig. 2. Representation of the chromosome used in AUC. The chromosome is a 1-D array of seven real numbers $[0, 1]$. Each position in the chromosome corresponds to one behavior component in the behavior-based controller described in Section II-B. The real number represents the probability of activating a behavior component whenever a waypoint is passed.

B. Artificial Stupidity

A game is more entertaining when an opponent's mistakes are intentional but plausible [21]. Artificial stupidity refers to the fine tuning of a game AI such that it provides the player with an entertaining experience by deliberately making mistakes. This also means that a game AI has to be overdesigned. That is, the game AI has to be able to defeat the player to begin with. Only when such a condition is satisfied can there be potential of deliberate handicapping. In the behavior-based AI, handicapping can be done by selectively activating or deactivating specific behavior components. This in turn makes the game more challenging and fun for the opposing player who now stands a chance at winning the game.

The adaptive controllers proposed in this paper adopt a similar approach to the above analogy. An overdesigned simulated car racing controller with a good set of game behavior components is first developed. Next, the adaptive controller estimates the competency of its opponent during the game and progressively selects a subset of behavior components to use for the remainder of the game so as to provide an engaging and satisfying game. The drawback of this approach is that the adaptive controller is only as good as its design and is unable to adapt to a player who is better than its full potential. A learning and adaptive game AI would be a natural future extension.

C. Adaptive Uni-Chromosome Controller

The AUC does not need to be trained offline. The training and adaptation process occurs in real time during the game. As implied by its name, AUC stores one chromosome which encodes seven real numbers $[0, 1]$, one for each of the seven behavior components, as shown in Fig. 2. Each real number represents the probability of activating a behavior component whenever a waypoint is passed. The expected behavior set encoded by this chromosome represents a "winning" strategy. In a sense, the chromosome models the proficiency level of the opponent by encoding a behavior set that is expected to be "good enough" to defeat him. It is assumed here (for simplicity but not necessarily realistic) that the complement of the probabilities of the "winning" strategy encoded by the chromosome would represent a potential "losing" strategy against the same opponent. The complement of an activation probability is calculated using

$$p_i' = 1 - p_i \tag{1}$$

where p_i is the probability of activation of behavior i encoded in the chromosome.

The chromosome is randomly initialized at the start of each game. When a waypoint is passed, the chromosome is updated by the following rules:

1) If AUC win
  for each behavior component ($i = 1$ to 7)
    if (rand() < myDist/(myDist + otherDist))
      win_i = (win_i + *sgn(behavior*$_i$*)* $\times l$) $\times m$;
2) If AUC lose
  for each behavior component ($i = 1$ to 7)
    if (rand() < otherDist/(myDist + otherDist))
      win_i = (win_i − *sgn(behavior*$_i$*)* $\times\ l$) $\times m$;

where *rand()* is a random number $[0, 1)$; *myDist* is the distance from the controller car to the destination at the previous update; *otherDist* is the distance from the opponent car to the destination at the previous update; win_i denotes the probability of activating the ith behavior component in the win chromosome for the next phase of the game; $behavior_i$ is the binary state of the ith behavior component before the update, 1 for activated and -1 for deactivated; l is the learning rate; and m is the mutation rate.

An important consideration here is the issue of credit assignment. For the car racing game, the relative distance of each car from the current waypoint determines the likelihood of reaching the waypoint first. For example, if the controller car is nearer to the destination than the opponent, then it is easier to win this waypoint, even with a weaker set of behavior components, by virtue of the closer proximity. Hence, this set of behavior components should be inherited by the chromosome with lower confidence. Conversely, if the controller car wins the waypoint when it is initially farther away from the destination, then that set of behavior components is demonstrated to be a winning strategy against this opponent and therefore it is inherited by the chromosome with higher confidence. In summary, the activation probability encoded in the chromosome is updated with the likelihood proportional to the relative distances of the cars to the destination. Finally, a mutation operator in the form of a Gaussian perturbation of mean zero is applied to each gene of the chromosome to introduce some diversity.

Each real number in the chromosome denotes the probability of activating the corresponding behavior component. Whenever a waypoint is passed, the AUC checks the new game state and chooses a set of behavior components for the next phase of the game according to the values encoded in its chromosome. Each time step in a game is classified into seven states based on the difference in score at that time step. From the perspective of the adaptive controller, the seven states are: losing by three points or more (g_{-3}); losing by two points (g_{-2}); losing by one point (g_{-1}); draw (g_0); winning by one point (g_1); winning by two points (g_2); and winning by three points or more (g_3). If the game state is g_{-3}, g_{-2}, g_{-1}, or g_0, a strategy is chosen based on the chromosome. This encourages the controller to try to win the next point if it is either losing or drawn in score. In cases of draw, the controller tries to go for a win so as to increase the tension in the game and to challenge the player to outperform it. If the game state is g_1, g_2, or g_3, the chromosome is complemented before the strategy is chosen.

D. Adaptive Duo-Chromosome Controller

The ADC is similar to AUC except that it does not assume the complement of an expected winning strategy to be a losing strategy. Instead, it maintains two sets of chromosomes, one winning chromosome and one losing chromosome, throughout the game. The update rules are modified as follows:

1) If ADC win
  for each behavior component ($i = 1$ to 7)
    if (rand() < myDist/(myDist + otherDist))
      win_i = (win_i + *sgn(behavior*$_i$*)* $\times l$) $\times m$;
2) If ADC lose
  for each behavior component ($i = 1$ to 7)
    if (rand() < otherDist/(myDist + otherDist))
      $lose_i$ = ($lose_i$ + *sgn(behavior*$_i$*)* $\times l$) $\times m$;

where *rand()* is a random number $[0, 1)$; *myDist* is the distance from the controller car to the destination at the previous update; *otherDist* is the distance from the opponent car to the destination at the previous update; win_i denotes the probability of activating the ith behavior component in the win chromosome for the next phase of the game; $lose_i$ denotes the probability of activating the ith behavior component in the lose chromosome for the next phase of the game; $behavior_i$ is the binary state of the ith behavior component before the update, 1 for activated and -1 for deactivated; l is the learning rate; and m is the mutation rate. The mutation operator is also applied to both chromosomes after the updating process. In this controller, whenever a waypoint is passed, ADC checks the new game state. If the game state is g_{-3}, g_{-2}, g_{-1}, or g_0, the win chromosome is used to generate the next behavior set. If the game state is g_1, g_2, or g_3, the lose chromosome is used instead.

IV. Static Controllers

The adaptive controllers were tested against the following static controllers of different driving characteristics used to simulate different players with different styles of play.

A. Heuristic Controller (HC)

The heuristic controller (HC) makes use of simple rules to collect waypoints in the game. It will steer in the direction of the current waypoint if its difference in heading exceeds a threshold value. It will accelerate if its speed is below its speed range or decelerate if above its speed range. The three parameters, speed limit, speed limit variance, and angle threshold, are optimized by evolution strategies, utilizing the plus selection scheme, population size of 50 and 200 generations. The fitness function is to maximize the number of waypoints collected against a simple hand designed HC. This controller does not have any predictive component so it ignores the existence of its opponent and always heads toward the current waypoint. The training fitness is shown in Fig. 3(a).

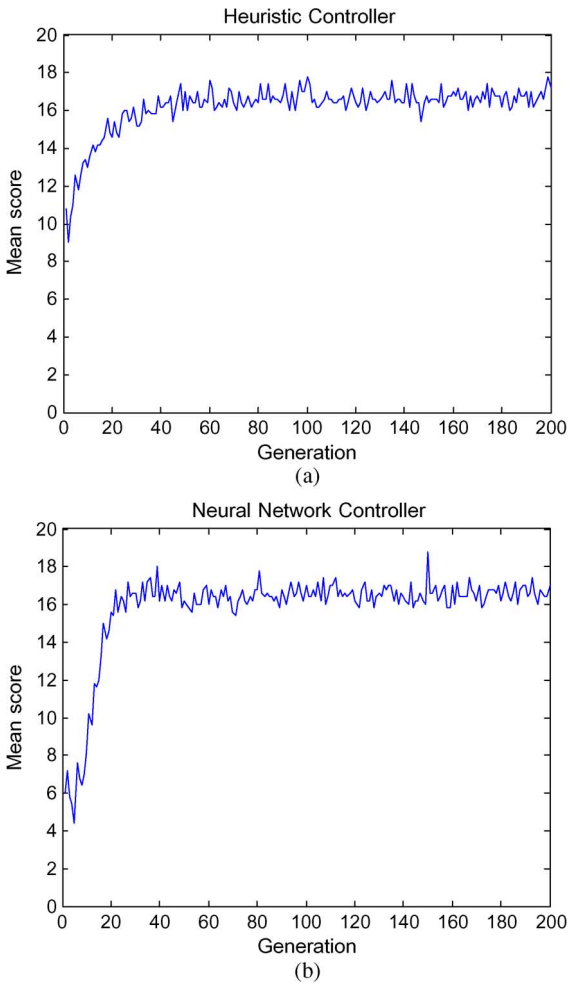

Fig. 3. Training fitness of (a) HC and (b) NNC.

B. Neural Network Controller (NNC)

A neural network of nine inputs, six hidden nodes, and two outputs is trained as a controller. The inputs are its own orientation, opponent orientation, own speed, angle to current waypoint, distance to current waypoint, angle to next waypoint, distance to next waypoint, angle to opponent, and distance to opponent. The outputs are steering control and driving control. Three additional parameters encoding the threshold to convert the neural network outputs to on/off controls are included in the training. The training conditions are identical to that of the HC. The neural network representation is capable of predictive properties, but this was not seen in the best evolved candidate. However, on visual observation, it drives in a smoother and more refined manner when compared to the HC. The training fitness is shown in Fig. 3(b).

C. Reverse Enabled Controller (RC)

The reverse enabled controller (RC) is a simplification of the behavior-based controller described earlier. The hyperbolic tangent speed regulator was deactivated and replaced with a hard speed limit of 5 pixels per time step. Only the reversing and direction switching compensation components were activated while all other driving and tactical components were deactivated. This controller ignores the opponent but makes good use of its ability to drive both forward and in reverse to collect waypoints, and recovers quickly from collisions.

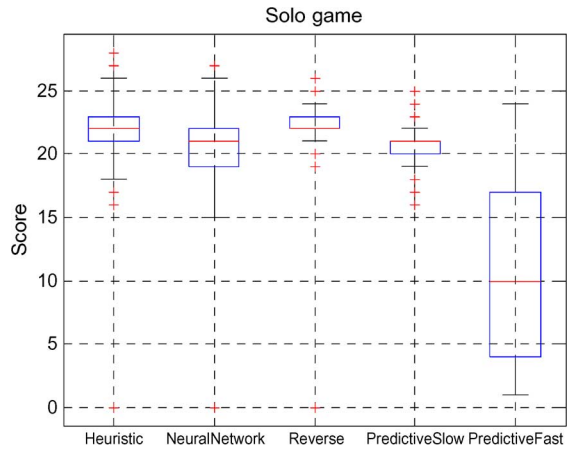

Fig. 4. Boxplot of the results for static controllers in solo games. An outlier at zero score indicates that the controller is unable to pass any waypoints at all, possibly due to it being trapped in a local orbit around the very first waypoint.

D. Predictive Slow Controller (PSC)

The predictive slow controller (PSC) is an extension of the HC with the addition of the waypoint prediction component from the set of tactical behavior components. The speed limit is set to 5 pixels per time step to simulate a relatively slower moving car compared to the HC. The lower speed limit will ensure that the car will not skid and hence is less likely to overshoot a waypoint.

E. Predictive Fast Controller (PFC)

The predictive fast controller (PFC) is a variation of the PSC, but with the speed limit set to 8 pixels per time step which is faster than the HC and PSC. The high speed limit means that the PFC will reach the waypoint faster but it also runs the risk of overshooting its destination. In addition, to prevent the fast moving car from getting trapped in orbit too frequently, a stopping mechanism is implemented to decelerate the car when it is sufficiently close to its destination.

F. Solo Game

The results of the solo run of the static controllers are presented in Fig. 4. The solo run gives an indication of the driving capabilities of the static controllers. Since there is no opponent vehicle in the solo run, the PSC and PFC are reduced to a variant of the HC but with different speed limits. It is observed that the first four controllers exhibit similar driving capabilities with NNC being slightly less consistent. PFC is the worst performing and most inconsistent controller with a low average score. This is due to its high speed limits which often results in skidding and hence it tends to overshoot a waypoint, wasting valuable time. However, PFC does not report any games with a zero score outlier, unlike HC, NNC, and RC. Zero score indicates that the

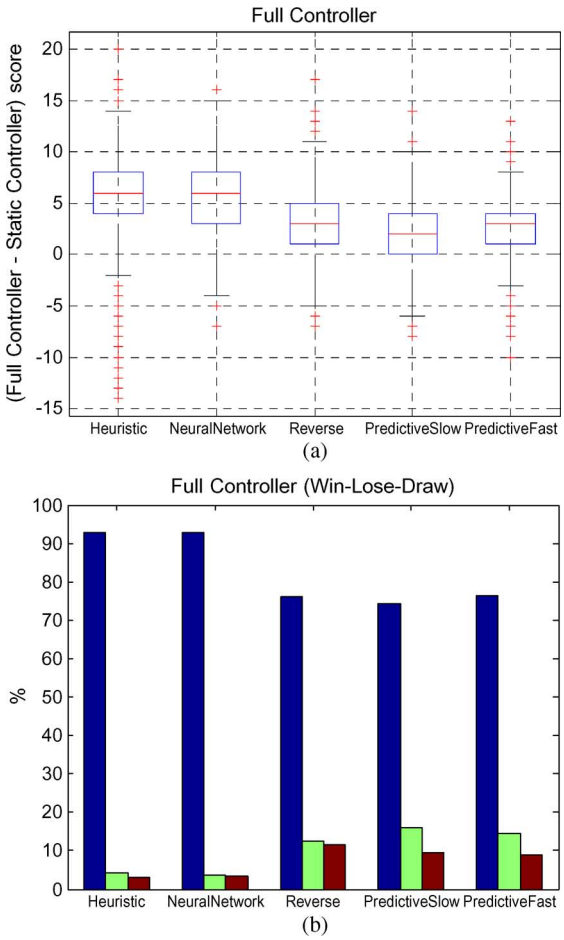

Fig. 5. FC against the five static controllers. (a) Boxplot showing the score differences. A positive score difference indicates that the FC won, while a negative score difference indicates that the opponent won. (b) Histogram of the game results. For each static controller, the blue, green, and red bars represent the percentage of games that the FC won, lost, and drew, respectively.

controller is stuck in orbit at the very first waypoint. This shows that a low speed limit (PSC) and a stopping mechanism (PFC) are effective in preventing orbiting.

V. Results and Discussions

The following experiments were carried out to evaluate the performance and effectiveness of the proposed adaptive algorithms. In all experiments, the results were obtained over $n = 5000$ games and each game lasted 1000 time steps.

A. Fully Activated Behaviors

The first step of our experiment is to establish the playing proficiency of the basis behavior-based controller to be used in the adaptive algorithms. We need to verify that the basis behavior-based controller is indeed an overdesign and hence possesses the potential to play even games against its opponents. The most competent controller is one with all behavior components permanently activated and represents the adaptive controller playing at full strength throughout the game. The full controller (FC) is played against each of the five static controllers described in Section IV. The results are presented in the form of a boxplot of the difference in score between the two players (i.e., the score of the FC minus the score of its opponent) in Fig. 5(a). A positive score difference indicates that the FC won a particular game while a negative score difference indicates that the opposing controller won the game. A score difference of zero would indicate a drawn game. The winning percentages from the perspective of the FC against each opponent are presented in Fig. 5(b). The blue, green, and red bars represent win, lose, and draw percentages, respectively.

It is observed that the FC is a very competent controller with positive median score differences against all its opponents. The lower quartile score differences are positive against four of its opponents and zero against only the PSC. This can also be observed in Fig. 5(b) where the FC obtained the lowest winning percentage of 74.42% against the PSC. However, it is clear that the FC is a very competent player against the static controllers. This makes the FC a suitable candidate to handicap itself during a game and to adapt to its opponent. The objective is to match an opponent in score (i.e., score difference should have a median value of zero) and also to match its winning and losing percentages.

B. Randomly Activated Behaviors

Besides having a suitable candidate for adaptation, the algorithm for adapting is also important such that it is effective against different types of opponents. A simple random algorithm is used in this experiment to demonstrate the need for guided learning in an adaptive algorithm. The random controller (RDC) operates by selecting a new set of behavior components at random to use every time a waypoint is passed. The RDC is played against each of the five static controllers described in Section IV and the results are presented in the form of a boxplot of the difference in score between the two players (i.e., the score of the RDC minus the score of its opponent) in Fig. 6(a). A positive score difference indicates that the RDC won a particular game and *vice versa*. A score difference of zero would indicate a drawn game. The winning percentages presented in Fig. 6(b) are from the perspective of the RDC. The blue, green, and red bars represent win, lose, and draw percentages, respectively.

It is observed from Fig. 6(a) that the median of the score differences are all negative. That is, the score of the RDC is lower than that of its opponent. This can be confirmed in Fig. 6(b) by observing that the RDC has a higher losing percentage than the winning percentage against all five opponents. This implies that a competent player, playing with random behavior components, is unable to consistently win against players of lower competency. Hence, the random use of effective behavior components does not equate to a good player. Furthermore, the choice of what combination of behavior components to use to match an opponent in a game needs to be guided.

C. Analysis of AUC

The AUC stores one chromosome which encodes seven real numbers, one for each of the seven behavior components, in the range $[0, 1]$. Each value represents the probability of activating a behavior component. The expected behavior set encoded by this chromosome represents a "winning" strategy. Whenever a waypoint is passed in the game, the chromosome is updated using the rules described in Section III-C and a new set of behavior components will be generated for use until the next waypoint is

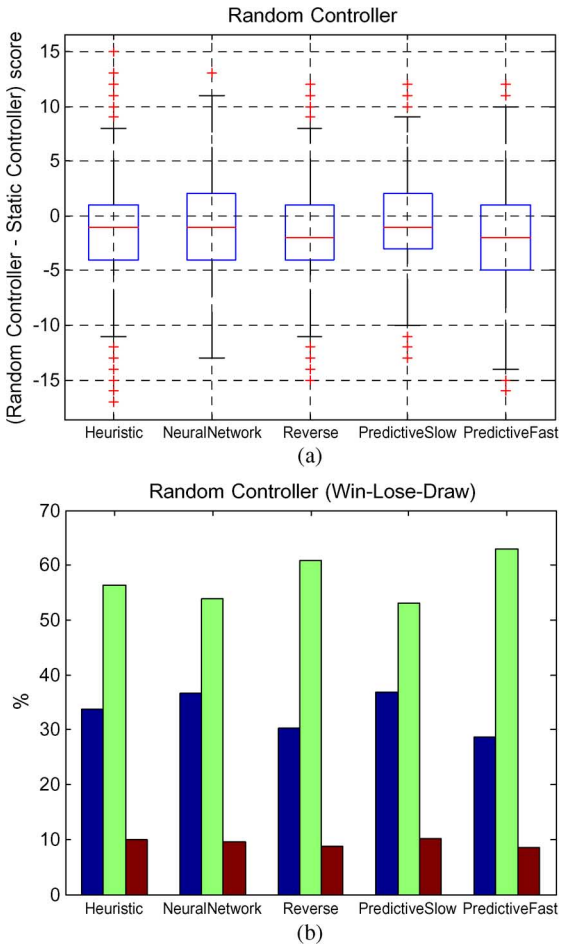

Fig. 6. RDC against the five static controllers. (a) Boxplot showing the score differences. A positive score difference indicates that the RDC won, while a negative score difference indicates that the opponent won. (b) Histogram of the game results. For each static controller, the blue, green, and red bars represent the percentage of games that the RDC won, lost, and drew, respectively.

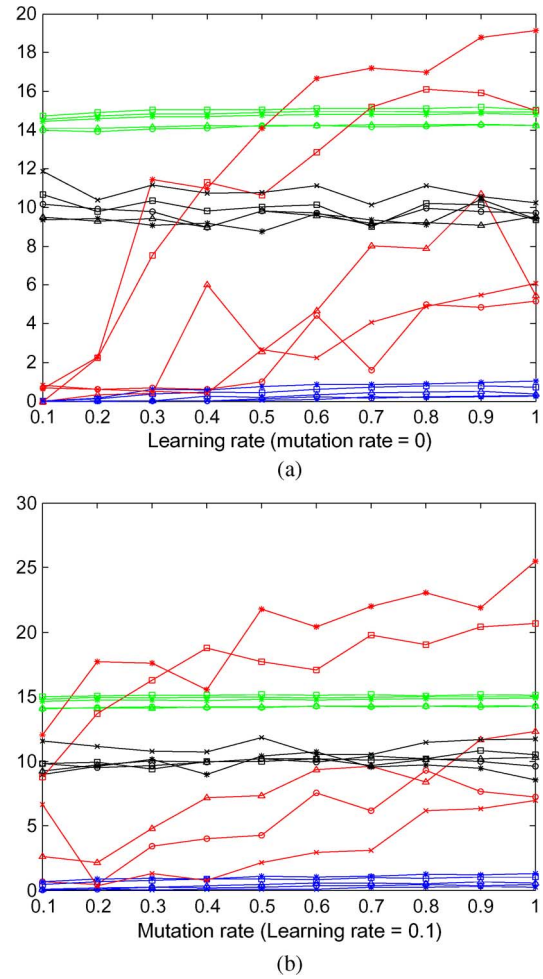

Fig. 7. Effects of varying (a) learning rate (mutation rate $= 0$) and (b) mutation rate (learning rate $= 0.1$) for AUC. The line colors red, black, blue, and green represent the difference in winning percentage $|W - L|$, number of draws D, difference between mean scores $|s1 - s2|$, and $\max(s1, s2)$, respectively. The markers triangle, circle, asterisk, square, and cross represent HC, NNC, RC, PSC, and PFC, respectively.

triggered. The effects of varying the learning and mutation rates are discussed in this section.

1) Effects of Varying Learning Rate: In this experiment, the mutation rate is set to zero and the learning rate is varied from 0.1 to 1.0 in steps of 0.1. The data of the mean scores, standard deviation, and winning percentages over $n = 5000$ games are omitted for conciseness. Instead, the results based on the two criteria described in Section III-A are graphically presented in Fig. 7(a). The difference in winning percentage $|W - L|$ (red line) should be minimal. The number of draws D (black line) should also be minimal because a high number of drawn games is deemed as more frustrating than fun. The difference between the mean scores $|s1 - s2|$ (blue line) should be minimal. The higher of the two scores $\max(s1, s2)$ (green line) should be maximal because a high average score indicates a competitive and fast paced game that is deemed to provide more satisfaction to the player.

It is observed from Fig. 7(a) that the general trend of increasing the learning rate is a gentle increase in mean score differences (blue) and also a large increase in winning percentage difference (red). This is because a large learning rate will quickly saturate the chromosome values to either 0 or 1. The resulting fluctuations in the chromosome values produce erratic behaviors that are unable to adapt and track the progress of its opponent during the game. At low learning rates, the score differences (blue) and winning percentage differences (red) are smaller and the AUC is able to match its opponent in both criteria. From numerical data, a learning rate of 0.1 obtained the best result for seven out of ten evaluation criteria (two evaluation criteria for each of five static controllers). It is also the dominant learning rate for three out of five static controllers, namely, HC, NNC, and PSC. Although the learning rate of 0.1 did not obtain the best result for either evaluation criteria against the PFC, the mean score difference of 0.01 and winning percentage difference of 0.84 are considered within acceptable range. Therefore, a learning rate of 0.1 is chosen as a good general rule of thumb that can be used in situations where opponents are varied and unknown. This value of learning rate will also be used as a default value in the experiment of varying mutation rate in Section V-C2.

It is also worth noting that from the numerical data, for $l > 0.5$, the mean score of the adaptive controller is higher than that

of the static controllers, but the winning percentage of the AUC is lower than that of the static controllers. This is likely caused by the AUC losing frequently by small margins but winning by large margins. This exemplifies that higher mean scores do not directly imply higher winning percentages.

2) Effects of Varying Mutation Rate: In this experiment, the learning rate is set to 0.1 by the observations in the previous section and the mutation rate is varied from 0.1 to 1.0 in steps of 0.1. The mutation rate controls the size of the standard deviation in a Gaussian perturbation of zero mean. The mutations are applied independently to each chromosome value after the learning rate is applied. The results are presented in Fig. 7(b).

It is observed from Fig. 7(b) that higher mutation rates tend to produce larger differences in mean score (blue) and winning percentage (red). Similar to the case of high learning rate, high mutation rate produces large fluctuations in the chromosome values, making the AUC overcompensate for its behavior components. This is analogous to noise being amplified by the differential component of a proportional–integral–derivative (PID) controller. From numerical data, the best performing mutation rate is 0.1 with seven out of ten best evaluation criteria. However, this result must be interpreted against the earlier result from varying the learning rate (i.e., $m = 0$). By comparison, the best results from $m = 0.1$ are worse (i.e., larger differences in mean score and winning percentage) compared to those from $m = 0$. This implies that the additional mutation operation may have introduced unnecessary divergence to the chromosome values, leading to poorer results. Therefore, in general, the mutation rate should be 0 for the AUC.

D. Analysis of ADC

In this section, the performance of the ADC will be assessed in terms of varying learning rate and varying mutation rate. The ADC differs from the AUC in that it does not make the assumption that the complement of a "winning" chromosome is a "losing" chromosome. Instead, it maintains two sets of chromosomes, each of which encodes seven real numbers in the range $[0, 1]$. One chromosome represents a "winning" behavior set while the other represents a "losing" behavior set. Each chromosome is updated independently when a waypoint is passed. However, the same learning and mutation rates are applied to both chromosomes.

1) Effects of Varying Learning Rate: In this experiment, the mutation rate is first set to zero and the learning rate is varied from 0.1 to 1.0 in steps of 0.1. The same learning rates are applied to both chromosomes of the ADC. The results are presented in Fig. 8(a). As earlier, the difference in winning percentage $|W - L|$ (red line) should be minimal, the number of draws D (black line) should be minimal, the difference between the mean scores $|s1 - s2|$ (blue line) should be minimal, and the higher of the two scores $\max(s1, s2)$ (green line) should be maximal.

It is observed in Fig. 8(a) that there are no trends with varying the learning rate. This is likely caused by the reduction of the frequency of update opportunities of each chromosome. The average number of updates during each game is the sum of the mean scores of the two players. With the ADC, only one of the two chromosomes is updated whenever a waypoint is passed,

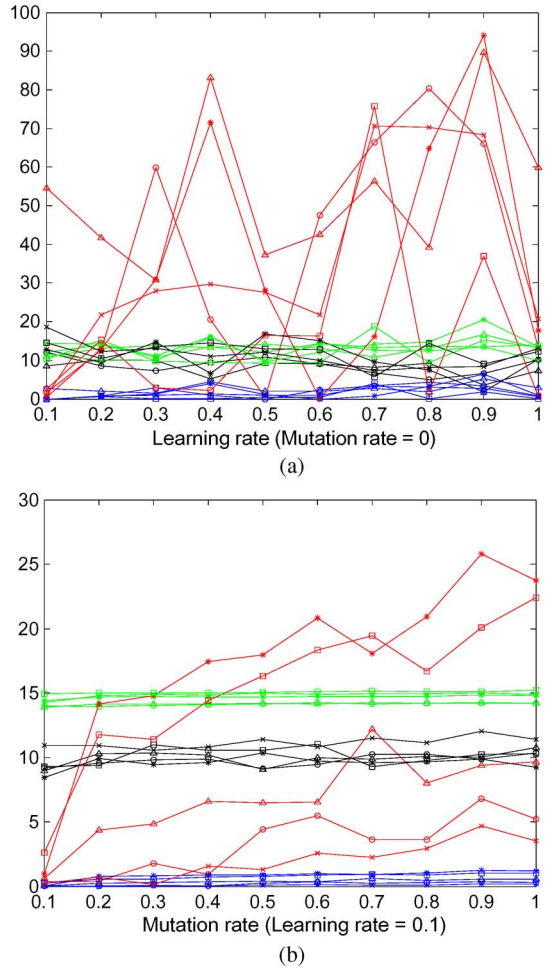

Fig. 8. Effects of varying (a) learning rate (mutation rate $= 0$) and (b) mutation rate (learning rate $= 0.1$) for ADC. The line colors red, black, blue, and green represent the difference in winning percentage $|W - L|$, number of draws D, difference between mean scores $|s1 - s2|$, and $\max(s1, s2)$, respectively. The markers triangle, circle, asterisk, square, and cross represent HC, NNC, RC, PSC, and PFC, respectively.

depending on whether the controller wins or loses the point. This means that, on average, each chromosome in the ADC is updated half as frequently as the chromosome in the AUC. The reduced updating frequency also reduced the effectiveness of the ADC to match its opponent in mean score and winning percentage. Nevertheless, from numerical data, a learning rate of 0.1 obtained the best result in five out of ten evaluation criteria. Therefore, $l = 0.1$ will be used as default value in the experiment of varying mutation rate in Section V-D2.

2) Effects of Varying Mutation Rate: In this experiment, the learning rate is set to 0.1 and the mutation rate is varied from 0.1 to 1.0 in steps of 0.1. The mutation rate controls the size of the standard deviation in a Gaussian perturbation of zero mean. The mutations are applied independently to each chromosome value after the learning rate is applied. The same mutation rate is applied to both chromosomes of the ADC. The results are presented in Fig. 8(b).

It is observed in Fig. 8(b) and numerical data that the mutation value of 0.1 obtained the best result in nine out of ten evaluation criteria. The results of the ADC improved greatly as

TABLE I
CUMULATIVE PERCENTAGE OF GAME ACCORDING TO SCORE DIFFERENCE

Score difference	AUC					ADC				
	Heuristic	Neural network	Reverse	Predictive slow	Predictive fast	Heuristic	Neural network	Reverse	Predictive slow	Predictive fast
0	0.0952	0.1016	0.0938	0.1068	0.1188	0.0896	0.0920	0.0846	0.0926	0.1094
≤1	0.2932	0.3078	0.2834	0.3106	0.3416	0.2586	0.2718	0.2596	0.2764	0.3168
≤2	0.4584	0.4826	0.4500	0.4792	0.5358	0.4322	0.4352	0.4362	0.4464	0.5050
≤3	0.6120	0.6278	0.5908	0.6272	0.6880	0.5838	0.5802	0.5806	0.5890	0.6540
≤4	0.7318	0.7422	0.7082	0.7404	0.8040	0.7038	0.7022	0.7024	0.7114	0.7808
≤5	0.8224	0.8336	0.8012	0.8286	0.8814	0.8010	0.7968	0.7938	0.8080	0.8634

a result of the introduction of the mutation operation. This is because the mutation operation offered more opportunities for the chromosome values to adapt compared to using the learning rate operation alone. The mean scores and winning percentages are similar to those of the AUC. Additionally, the ADC ($l = 0.1$, $m = 0.1$, nine of ten evaluation criteria) is able to produce more consistent results compared to the AUC ($l = 0.1$, $m = 0$, seven of ten evaluation criteria) in response to varied and unknown opponents. The disadvantage is that the ADC has a larger memory footprint.

E. Score Difference Distribution

Both the AUC and the ADC, using the optimized parameters, have been demonstrated to be effective in matching their opponent in terms of mean score difference and winning percentage difference. In this section, the distribution of the difference in score in each of the $n = 5000$ games will be further analyzed. The following analysis will be divided into two sections, namely, the overall distribution of score differences and the distribution of the occurrence of the score differences.

1) Distribution of Score Difference: The significance of analyzing the distribution of score differences is to investigate the effects of the adaptive controller on the game experience of its opponents. A game experience is considered satisfying or entertaining when it is difficult to defeat. This idea can also be extended to say that a game experience is considered satisfying or entertaining when it is won or lost by a small margin. In the context of the simulated car racing game, this can be interpreted as a small score difference between the two competing players. From real-world user experience, a game is subconsciously considered won or lost by a player when a score difference of more than 5 is observed during the game. The typical score of a player for one game is around 13–15 points. Therefore, if the end game score difference is 4 or less, it can be said that the player is entertained during game. The boxplot of the score differences of the AUC with optimized parameters against the five static opponents is presented in Fig. 9. The results for the ADC (not shown) are similar. The histogram of the score differences of the adaptive controllers against the PFC is presented in Fig. 10. The results when playing against the other static controllers are similar and not shown here. The number of game results that fall within a specific score difference is summarized in Table I as a percentage of total games played.

It is observed that the median values of both the AUC and the ADC against all their opponents are zero. It is also observed from Fig. 10 that, by the near symmetry of the histogram, their mean values are very close to zero. These imply that both adaptive controllers are able to match their opponents in terms of mean score difference and winning percentage difference. The upper and lower quartiles of both adaptive controllers are 3 and -3, respectively, for HC, NNC, RC, and PSC. It is 2 and -2, respectively, for the PFC. It is also observed from Table I that a minimum of 70.22% of the game results between the adaptive and static controllers have a score difference of 4 or less. This indicates that the opponent is entertained in at least 70.22% of the games.

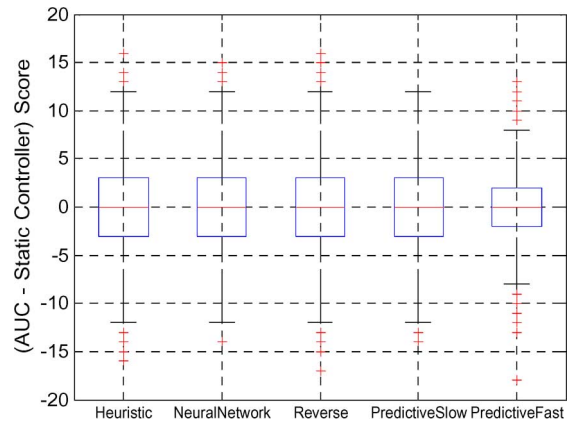

Fig. 9. Boxplot of the results from playing the AUC against the five static controllers. The results are shown in terms of score differences between the AUC and its opponent. A positive score difference indicates that the AUC won, while a negative score difference indicates that the opponent won.

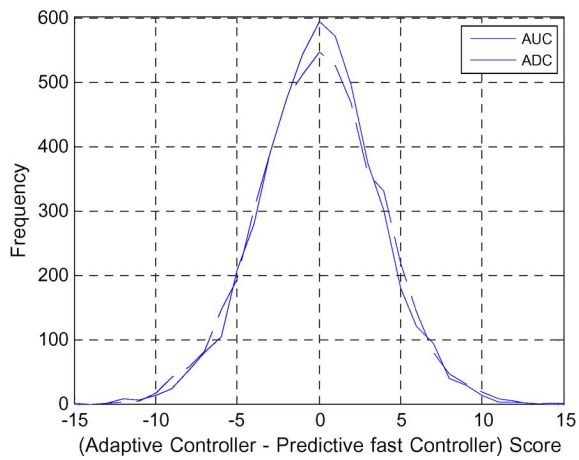

Fig. 10. Histogram of the score difference of the adaptive controllers against the PFC. A positive score difference indicates that the adaptive controller won, while a negative score difference indicates that the static controller won.

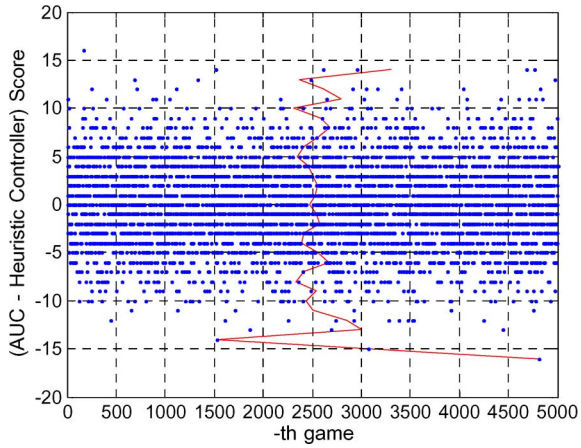

Fig. 11. A sample diagram of 5000 games between the AUC and the HC. The dots represent individual data points of the score difference plotted against the game count. The line represents the mean occurrence of the score difference.

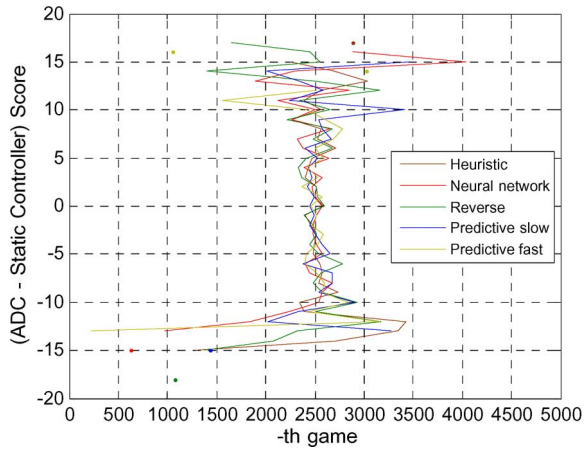

Fig. 12. Plot of the score difference between the ADC versus static controllers against the game count. The lines represent the mean occurrence of the score difference against the static controllers. Mean occurrences near the 2500th game indicate that the score difference represented in the vertical axis is evenly distributed across the total number of games played.

It is observed from Fig. 10 that the ADC tends to have a lower number of drawn games compared to the AUC. This is a desirable effect as drawn games are deemed to be more frustrating than fun. Hence, the ADC has the advantage of being able to consistently produce a low frequency of drawn games against varied and unknown opponents.

2) Distribution of the Occurrence of the Score Difference: The purpose of investigating the distribution of the occurrence of the score difference is to verify that each score difference is evenly distributed over the $n = 5000$ games. That is, the adaptive controller should win by two points, as well as lose by two points, regularly over the 5000 sequential games. As an extreme example, the opposite would be to win the first 2500 games by two points and lose the last 2500 games by two points. In this example, the mean score difference and the winning percentage difference are zero but the opposing player will feel dissatisfied by losing the first 2500 games. A sample diagram of the 5000 games between the AUC and the HC is shown in Fig. 11.

The horizontal axis represents the game count while the vertical axis represents the score difference. Each dot represents the outcome of the game while the line represents the mean occurrence of the score difference on the vertical axis. It is desirable that the mean occurrence of each score difference is around the 2500th game (i.e., vertically along the center line). This would imply that the particular score difference is evenly distributed. It is observed from Fig. 11 that the mean occurrence values for score differences of more than 5 tend to diverge away from the center line. This is due to a low frequency of these score differences as 82.24% of the games occur with a score difference of 5 and less. As such, only score differences of 5 and lower are considered for this analysis.

The results of all the games for the ADC against the five static controllers are plotted together in Fig. 12. The individual game outcomes are left out to make the diagram more reader friendly. Only the mean value lines are plotted. The results for the AUC are similar and not shown here. It is observed that both adaptive controllers have consistent and near zero mean occurrences for score differences in the range of -5 to 5. This indicates that both adaptive controllers are able to evenly distribute varying score differences across a long run of sequential games. This helps to keep the opposing player interested in the game by uniformly winning and losing.

F. Behavior Activation Probability Distribution

In this section, the final values of the behavior component activation probability that is encoded in the chromosomes of the adaptive controllers will be discussed. The objective of this discussion is to identify any general trends or preferences in behavior components when the adaptive controllers are playing against varied opponents.

1) Analysis of AUC: The AUC contains one chromosome which encodes seven real numbers, one for each of the seven behavior components. Each value represents the probability of activating a specific behavior component. The expected behavior set encoded by this chromosome represents a "winning" strategy while its complement is assumed to be a "losing" strategy. The boxplots and histograms of the final chromosome values against PFC are presented in Fig. 13. The results when playing against the other static controllers are similar and not shown here. The line in the boxplot connects the means of each chromosome position. The histogram consists of ten bins with an interval of 0.1 from 0 to 1.

Some general trends regarding the final chromosome values encoded by the AUC are observed from Fig. 13. The reversing and direction switching behavior components are selected with high probabilities against all its opponents, indicating that these two behavior components are important behavior components to choose if the AUC wants to express a winning strategy. The tight angle turning and time wasting behavior components have means and medians of around 0.5 against all opponents. This implies that these two behavior components are not as significant in deciding whether a strategy is a winning one. The blocking behavior component is selected with the lowest probability among the other behavior components. This is because blocking an opponent during a game is a defensive behavior to prevent the opponent from getting a point rather than to gain a point for itself. Hence, the AUC generally assigns

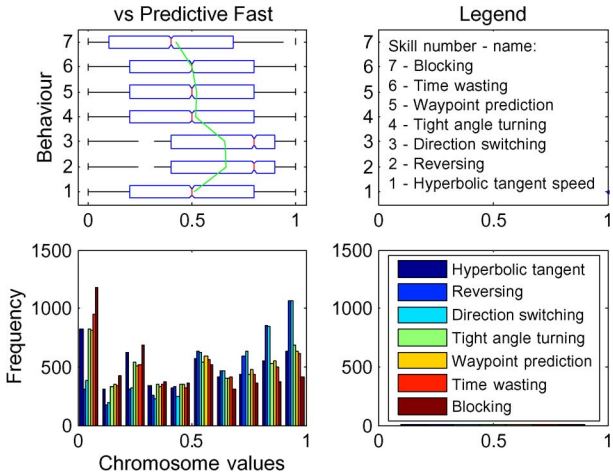

Fig. 13. Boxplot (top left) and histogram (bottom left) of final chromosome values of the AUC against the PFC. The vertical axis of the boxplot represents the behavior components while the horizontal axis represents the chromosome values in the range $[0, 1]$.

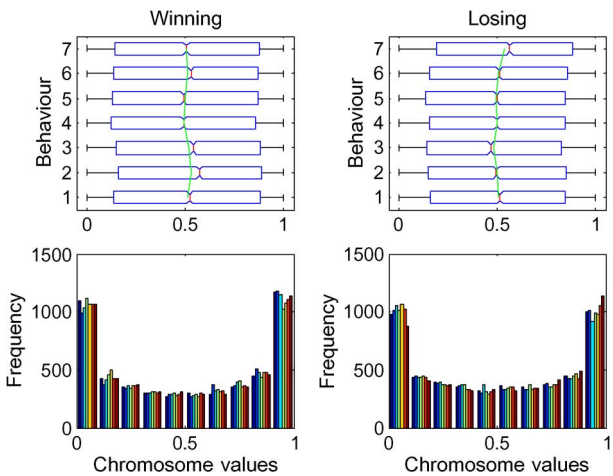

Fig. 14. Boxplot and histogram of final chromosome values of the ADC against the PFC. The winning and losing chromosomes are shown on the left and right columns, respectively. The vertical axis of the boxplot represents the behavior components while the horizontal axis represents the chromosome values in the range $[0, 1]$. Legend is the same as that in Fig. 13.

a lower probability of activation for this behavior component in its chromosome which encodes a winning strategy. This demonstrates an advantage of the proposed adaptive algorithm, in that it is able to automatically select a suitable subset of winning behavior components via its chromosome.

2) Analysis of ADC: The ADC consists of two chromosomes instead of one; each encodes seven real numbers. One of the chromosomes represents a "winning" strategy while the other represents a "losing" strategy. The boxplots and histograms of the final values of both chromosomes for the ADC against the PFC are presented in Fig. 14. The results when playing against the other static controllers are similar and not shown here. The line in the boxplot connects the mean values of each chromosome position. The histogram consists of ten bins with an interval of 0.1 from 0 to 1.

It is observed from the histogram in Fig. 14 that both winning and losing chromosomes tend to produce high frequencies for the two bin values nearest to 0 and 1. This is likely due to the positive reinforcement nature of the update rules used in the ADC update rules. The behavior components that resulted in the winning of a waypoint were updated in the winning chromosome only and not in the losing chromosome. Conversely, behavior components that resulted in the losing of a waypoint were updated solely in the losing chromosome. This resulted in a high frequency of chromosomes taking values at the extremities. The shape of the boxplot of the winning chromosome resembles that of the chromosome from the AUC. This is expected as the chromosome from the AUC encodes a winning strategy as well. There are no observable trends in the losing chromosome. This may indicate that there are many possible combinations of losing behavior components and that the algorithm learns a different one each time.

VI. Conclusion

Two adaptive algorithms, namely, AUC and ADC, have been introduced in this paper to enhance player satisfaction. The effects of varying the learning rate and mutation rate were investigated for both algorithms and a general rule of thumb for the selection of these two parameters was put forward. The distribution of the score difference has been examined and both algorithms were able to achieve a score difference of 4 or lower for a minimum 70.22% of the games. The occurrence of wins and losses was also well distributed over the sequence of consecutive games. It has also been observed that while the AUC had a smaller memory footprint, the ADC is able to maintain a lower number of drawn games which may help to reduce player frustration. In examining the final values of the chromosomes, it was found that the adaptive algorithms select different combinations of behavior components to cope with different opponents, although the reversing and direction switching behavior components were observed to be more prominent in winning chromosomes. Both proposed adaptive algorithms were able to automatically learn suitable sets of behavior components to match the different opponents in terms of mean score and winning percentage. Also, both proposed adaptive algorithms were able to generalize well to a variety of opponents. As an extension to this work, an idea is to expand the game to involve three players consisting of two game AIs and one human player. It would be interesting to examine how two game AIs could cooperate to further enhance the satisfaction of the human player.

References

[1] A. Nareyek, "Game AI is dead. Long live game AI!," *IEEE Intell. Syst.*, vol. 22, no. 1, pp. 9–11, Jan.-Feb. 2007.
[2] K. Forbus and J. Laird, "AI and the entertainment industry," *IEEE Intell. Syst.*, vol. 17, no. 4, pp. 15–16, Jul.-Aug. 2002.
[3] J. Schaeffer, "A gamut of games," *Artif. Intell. Mag.*, vol. 22, no. 3, pp. 29–46, 2001.
[4] J.-B. Hoock, C.-S. Lee, A. Rimmel, F. Teytaud, M.-H. Wang, and O. Teytaud, "Intelligent agents for the game of go," *IEEE Comput. Intell. Mag.*, vol. 5, no. 4, pp. 28–42, Nov. 2010.
[5] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," in *Proc. Workshop Adaptive Approaches Optim. Player Satisfaction Comput. Phys. Games*, 2006, p. 70.
[6] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalized content creation for racing games," in *Proc. IEEE Symp. Comput. Intell. Games*, 2007, pp. 252–259.
[7] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Mach. Learn.*, vol. 63, no. 3, pp. 217–248, 2006.

[8] M. Bergsma and P. Spronck, "Adaptive intelligence for turn-based strategy games," in *Proc. Belgian-Dutch Artif. Intell. Conf.*, 2008, pp. 17–24.
[9] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen, "Interactive storytelling: A player modelling approach," in *Proc. Artif. Intell. Interactive Digit. Entertain.*, 2007, pp. 43–48.
[10] B. D. Bryant and R. Miikkulainen, "Neuroevolution for adaptive teams," in *Proc. IEEE Congr. Evol. Comput.*, 2003, vol. 3, pp. 2194–2201.
[11] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 653–668, Dec. 2005.
[12] G. N. Yannakakis, "AI in computer games: Generating interesting interactive opponents by the use of evolutionary computation," Ph.D. dissertation, Schl. Inf., Inst. Perception, Action, Behaviour, Univ. Edinburgh, Edinburgh, U.K., 2005.
[13] J. Togelius and S. M. Lucas, "Evolving controllers for simulated car racing," in *Proc. IEEE Congr. Evol. Comput.*, 2005, vol. 2, pp. 1906–1913.
[14] J. Togelius and S. M. Lucas, "IEEE CEC 2007 Car Racing Competition," Aug. 18, 2008 [Online]. Available: http://julian.togelius.com/cec2007competition/
[15] H. Tang, C. H. Tan, K. C. Tan, and A. Tay, "Neural network versus behavior based approach in simulated car racing," in *Proc. IEEE Workshop Evolving Self-Developing Intell. Syst.*, 2009, pp. 58–65.
[16] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. 2, no. 1, pp. 14–23, Mar. 1986.
[17] M. Ramsey, "Designing a multi-tier AI framework," in *AI Game Programming Wisdom*. Newton Center, MA: Charles River Media, 2003, vol. 2.
[18] M. Buro and T. Furtak, "RTS games as a test-bed for real-time AI research," in *Proc. 7th Joint Conf. Inf. Sci.*, 2003, pp. 481–484.
[19] B. Scott, "The illusion of intelligence," in *AI Game Programming Wisdom*. Newton Center, MA: Charles River Media, 2002, pp. 16–20.
[20] T. W. Malone, "What makes things fun to learn? Heuristics for designing instructional computer games," in *Proc. 3rd ACM SIGSMALL Symp./1st SIGPC Symp. Small Syst.*, 1980, pp. 162–169.
[21] L. Liden, "Artificial stupidity: The art of intentional mistakes," in *AI Game Programming Wisdom*. Newton Center, MA: Charles River Media, 2004, vol. 2.
[22] K. L. Tan, C. H. Tan, K. C. Tan, and A. Tay, "Adaptive game AI for Gomoku," in *Proc. 4th Conf. Autonom. Robots Agents*, 2009, pp. 507–512.
[23] J. H. Kim, Y. H. Kim, S. H. Choi, and I. W. Park, "Evolutionary multiobjective optimization in robot soccer system for education," *IEEE Comput. Intell. Mag.*, vol. 4, no. 1, pp. 31–41, Feb. 2009.
[24] P. Durr, C. Mattiussi, and D. Floreano, "Genetic representation and evolvability of modular neural controllers," *IEEE Comput. Intell. Mag.*, vol. 5, no. 3, pp. 10–19, Aug. 2010.
[25] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Automatic computer game balancing: A reinforcement learning approach," in *Proc. 4th Int. Conf. Autonom. Agents Multiagent Syst.*, 2005, pp. 1111–1112.
[26] M. O. Riedl and A. Stern, "Believable agents and intelligent story adaptation for interactive storytelling," in *Proc. 3rd Int. Conf. Technol. Interactive Digit. Storytelling Entertain.*, 2006, pp. 1–12.
[27] H. Y. Quek, K. C. Tan, and A. Tay, "Public goods provision: An evolutionary game theoretic study under asymmetric information," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 2, pp. 105–120, Jun. 2009.
[28] G. N. Yannakakis and J. Hallam, "Real-time game adaptation for optimizing player satisfaction," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 2, pp. 121–133, Jun. 2009.
[29] J. Hagelback and S. J. Johansson, "Measuring player experience on runtime dynamic difficulty scaling in an RTS game," in *Proc. 5th Int. Conf. Comput. Intell. Games*, 2009, pp. 46–52.
[30] C. H. Tan, K. C. Tan, and A. Tay, "Computationally efficient behavior based controller for real time car racing simulation," *Expert Syst. Appl.*, vol. 37, no. 7, pp. 4850–4859, 2010.
[31] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992.
[32] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Adv. Neural Inf. Process. Syst.*, vol. 12, pp. 1057–1063, 2000.
[33] H. Wang, Y. Gao, and X. Chen, "RL-DOT: A reinforcement learning NPC team for playing domination games," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 1, pp. 17–26, Mar. 2010.
[34] H. Barber and D. Kudenko, "Generation of adaptive dilemma-based interactive narratives," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 4, pp. 309–326, Dec. 2009.
[35] P. G. Balaji and D. Srinivasan, "Multi-agent system in urban traffic signal control," *IEEE Comput. Intell. Mag.*, vol. 5, no. 4, pp. 43–51, Nov. 2010.
[36] M. Csikszentmihályi, *Flow: The Psychology of Optimal Experience*. New York: Harper Collins, 1990.
[37] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Difficulty scaling of game AI," in *Proc. 5th Int. Conf. Intell. Games Simul.*, 2004, pp. 33–37.

Chin Hiong Tan received the B.Eng. degree from the Department of Electrical and Computer Engineering, National University of Singapore, Singapore, in 2006 and the Ph.D. degree from the Centre for Intelligent Control, National University of Singapore, Singapore, in 2010.

He is currently a Scientist at the Institute for Infocomm Research, Agency for Science Technology and Research (A*STAR), Singapore. His research interests include evolutionary computation and neural networks for enhancing player satisfaction in games, and computational cognitive memory models of the hippocampus.

Kay Chen Tan received the B.Eng. (first class honors) and Ph.D. degrees from the University of Glasgow, Glasgow, Scotland, in 1994 and 1997, respectively.

He is currently an Associate Professor at the Department of Electrical and Computer Engineering, National University of Singapore (NUS), Singapore. He has published over 200 journal and conference papers and coauthored five books.

Dr. Tan has been invited to be a keynote/invited speaker for 21 international conferences, and served in the international program committee for over 100 conferences. He was involved in the organizing committee for over 30 international conferences. He is currently a Distinguished Lecturer of the IEEE Computational Intelligence Society and the Editor-in-Chief of the IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE. He also serves as an Associate Editor/Editorial Board member of 15 international journals, such as the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, *Evolutionary Computation* (MIT Press), *European Journal of Operational Research*, *Journal of Scheduling*, and *International Journal of Systems Science*. He is a Fellow of the NUS Teaching Academic.

Arthur Tay received the B.Eng. (first class honors) and Ph.D. degrees from the Department of Electrical and Computer Engineering, National University of Singapore (NUS), Singapore, in 1995 and 1998, respectively.

He was a Visiting Scholar with the Information System Laboratory, Stanford University, Stanford, CA, from 1998 to 2000. He then joined the Department of Electrical & Computer Engineering, NUS, where he is currently an Associate Professor. His current research interest is in the field of control engineering with applications to semiconductor manufacturing and biomedical technology.