ENHANCING PLAYER EXPERIENCE IN COMPUTER GAMES: A COMPUTATIONAL INTELLIGENCE APPROACH

TAN CHIN HIONG
B.Eng (Hons., 1st Class), NUS

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

2010

Summary

Gaming is by definition an interactive experience that often involves the human player interacting with the non-player characters in the game which are in turn controlled by the game artificial intelligence. Research in game AI has traditionally been focused on improving its competency. However, a competent game AI does not directly correlate to the satisfaction and entertainment value experienced by the human player. This thesis focuses on addressing two key issues of game AI affecting the player experience, namely adaptability and believability, in real time computer games from a computational intelligence perspective.

The nature of real time computer games requires that the game AI be computationally efficient in addition to being competent in the game. This thesis starts off by proposing a hybrid evolutionary behaviour-based design framework that combines the good response time of behaviour-based systems and the search capabilities of evolutionary algorithms. The result is a scalable framework where new behaviours can be easily introduced. This lays the groundwork for investigations into enhancing the player experience.

Two adaptive algorithms are built upon the proposed framework to address the issue of adaptability in games. The two proposed adaptive algorithms draw inspirations from reinforcement learning and evolutionary algorithms to dynamically scale the difficulty of the game AI while the game is being played such that offline training is not necessary. Such an adaptive system has the potential to customize a personalized experience that grows together with the human player.

The game AI framework is also augmented by the introduction of evolved sensor noise in order to induce game agents with believable movement behaviours. Furthermore, the action histogram and action sequence histogram are explored as a means to quantify the believability of the game agent’s movements. A multi-objective optimization approach is then used to improve the believability of the game agent without degrading its performance and the results are verified in a user study. Improving the believability of game agents has the potential to maintain the suspension of disbelief and increase immersion in the game environment.

List of Publications

Journals

Tan, C. H., Tan, K. C. and Tay, A., "Computationally Efficient Behaviour Based Controller for Real Time Car Racing Simulation", Expert Systems with Applications, vol. 37, no. 7, pp. 4850-4859, 2010.

Tan, C. H., Ramanathan, K., Guan, S. U. and Bao, C., "Recursive Hybrid Decomposition with Reduced Pattern Training", International Journal of Hybrid Intelligent Systems, vol. 6, no. 3, pp. 135-146, 2009.

Togelius, J., Lucas, S., Ho, D. T., Garibaldi, J. M., Nakashima, T., Tan, C. H., Elhanany, I., Berant, S., Hingston, P., MacCallum, R. M., Haferlach, T., Gowrisankar, A. and Burrow, P., "The 2007 IEEE CEC simulated car racing competition", Genetic Programming and Evolvable Machines, vol. 9, no. 4, pp. 295-329, 2008.

Tan, C. H., Tan, K. C. and Tay, A., "Dynamic Game Difficulty Scaling using Adaptive Behavioural Based AI", IEEE Transactions on Computational Intelligence and AI in Games, *accepted*.

Tan, C. H., Tan, K. C. and Tay, A., "Evolving Believable Behaviour in Games using Sensor Noise and Action Histogram", Evolutionary Computation, *submitted*.

Conference papers

Tang, H., Tan, C. H., Tan, K. C. and Tay, A., "Neural Network versus Behaviour Based Approach in Simulated Car Racing", Proceedings of IEEE Workshop on Evolving and Self-Developing Intelligent Systems, pp. 58-65, 2009.

Tan, K. L., Tan, C. H., Tan, K. C. and Tay, A., "Adaptive Game AI for Gomoku", Proceedings of the Fourth International Conference on Autonomous Robots and Agents, pp. 507-512, 2009.

Tan, C. H., Ang, J. H., Tan, K. C. and Tay, A., “Online Adaptive Controller for Simulated Car Racing”, Proceedings of IEEE Congress on Evolutionary Computation, pp. 2239-2245, 2008.

Ang, J. H., Teoh, E. J., Tan, C. H., Goh, K. C. and Tan, K. C., “Dimension Reduction using Evolutionary Support Vector Machines”, Proceedings of IEEE Congress on Evolutionary Computation, pp. 3635-3642, 2008.

Tan, C. H., Goh, C. K., Tan, K. C. and Tay, A., “A Cooperative Coevolutionary Algorithm for Multiobjective Particle Swarm Optimization”, Proceedings of IEEE Congress on Evolutionary Computation, pp. 3180-3186, 2007.

Acknowledgements

First and foremost, I would like to thank my Ph.D. supervisor, Associate Professor Tan Kay Chen for giving me the opportunity to pursue research in the field of computational intelligence. His indispensable guidance and kind words of encouragement kept me motivated and on track throughout my candidature. I would also like to thank my co-supervisor, Associate Professor Arthur Tay for his support in both my research and my participation in the ECE outreach program.

I would also like to extend my gratitude to Sara, Hengwei and Chee Siong for giving me the logistical support during my time at the lab; and the outreach staff Henry and Marsita for making my outreach experience one filled with fun and enjoyment.

I am also grateful to my fellow labmates at the Control and Simulation lab for making my four years of Ph.D. life full of fond memories: Chi Keong for always providing novel and interesting research suggestions; Dasheng for always being there when it is time to Bang!; Eujin for our numerous late night journeys to the bus interchange; Brian for literally bringing us round our sunny island in search of food and games; Chiam for bringing BS to the group; Chun Yew for always organizing our four player incomplete information zero sum set collection excursions; Han Yang for sharing with me his enthusiasm for film and traveling; Teck Wee (from the lab upstairs) for teaching me so much about photography during our trip to Hong Kong; Vui Ann for his ever jovial presence; Calvin for giving me new perspectives on a teaching career; and Jun

Yong for helping to rearrange all the furniture when our work space underwent renovations during the holidays.

Last but not least, I wish to thank my parents and sister for all their love and support. I wish to especially thank my wife, Juney, for going on this journey with me, for together building a family we can call our own, for giving birth to our wonderful daughter, for always being there. Finally, I wish to thank my 6 month old daughter, Yurou, for melting my heart everyday with her toothless baby grin. Kyaa~ ☺

Table of Contents

List of Tables

List of Figures

Chapter One

1 Introduction

Computer games play many roles in the society today. For example, military simulations in the form of war-games are used in military training. Management simulations and economic simulations are also becoming valuable training tools in their industries. Educational games have gained widespread acceptance for enhancing the learning experience of pre-school children. However, the most prominent role of computer games is still one as a form of entertainment.

The computer game industry has seen tremendous growth in the recent decade. According to the Entertainment Software Association, the sales of computer games in the U.S. grew from 2.6 billion U.S. dollars in 1996 to 7.6 billion U.S. dollars in 2004 to 11.7 billion U.S. dollars in 2008 [44]. Coupled with the constant broadening of gamer demographics in both age and gender as a result of casual gaming, the computer game industry has the potential to reach out to a widening range of audiences and continue its growth in the near future.

The quality of computer games, and hence its success, is directly related to their entertainment value [182]. Traditionally, game developers competed with one another in terms of a game’s graphical presentation and visual effects. However, in recent years, as graphics improvements begins to saturate, game developers are attempting to compete by offering better gameplay experiences through other means. Game artificial intelligence (AI), being an essential part of a gameplay experience, has emerged as an important selling point of games [49].

Gaming is inherently an interactive experience that involves the human player interacting with the non-player characters (NPC) in the game which are in turn controlled by the game AI. Research in game AI has traditionally been focused on improving its competency. However, a competent game AI does not directly correlate to the satisfaction and entertainment value experienced by the human player. The player experience also depends on other factors such as the suitability of the challenge provided, the amount of curiosity invoked, the level of rationality presented by the NPC, amongst others. This thesis focuses on the use of computational intelligence techniques on two key issues of game AI affecting the player experience, namely adaptability and believability.

1.1 Game AI and computational intelligence

Artificial intelligence (AI), as explained by one of the founders of the field, John McCarthy, is the science and engineering of making intelligent machines, especially intelligent computer programs. AI is derived from a branch of computer science that seeks to create intelligence for machines. An intelligence machine or agent can be seen as an embodied system that is able to perceive its environment and execute actions or sequence of actions that fulfills or brings it closer to its desired outcome. The study of AI encompasses areas such as reasoning, planning and scheduling, speech and facial recognition, natural language, behavioural learning and adaptation. Its applications are deeply embedded in day to day living, more so than most people realize. These systems range from directing road traffic, managing public transportation schedules and making weather predictions to interactive gaming, filtering spam e-mails and returning relevant results for an Internet search.

The goal for AI that researchers set for themselves is an ambitious one, one that would pass the Turing test described by Alan Turing in 1950 [183]. A machine is said to pass the test if a human judge cannot reliably distinguish whether it is a human or machine in a natural language conversation. Livingstone also discussed the Turing test in the context of games [85]. Today, AI research still has not produced a machine with sufficient common sense to describe a static scene, but it did develop Deep Blue, the IBM supercomputer that defeated the human chess champion in 1997 [74]. Common sense, ironically, turns out to be a difficult challenge in AI research. This led to the paradigm shift from mimicking human intelligence to advancing expert systems in specific focused applications. Currently, AI technology is used by search engines to organize data, helping doctors with diagnosis and treatment, and employed by police for fraud detection. Computer games, nonetheless, is still an ideal platform for AI research [28].

Game AI today is an interdisciplinary field consisting of knowledge based systems, machine learning, multi-agent systems, computer graphics, animation and data structures. Game AI is about creating the illusion of human behaviour. It needs to be smart to a certain extent, make unpredictable but rational decisions. A NPC controlled by the game AI needs to display emotional influences and make use of body language to communicate emotions to the player.

In order to create the illusion of human behaviour, the game AI is not allowed to cheat obviously. Cheating methods such as allocating more resources, neglecting speed limits, and switching off fog-of-war for computer controlled opponents had been commonly employ in game AI. But these types of obvious cheating are easily detected by the human player and generally degrade the gameplay experience. In other words, sensory honesty is a fundamental requirement for game agents [76]. In addition, game AI should be not display obviously stupid behaviour such as being stuck in a corner, or jumping out of a window under no threat. More importantly, game AI that exhibit self-correction, learning from experience and creative maneuvers will improve their perceived intelligence. It should also be noted that, in general, game AI has the inherent advantage of not being required to manipulate the graphical user interface (GUI), and is therefore faster when it comes to issuing game commands to the game engine.

Game designers of early computer games have already acknowledged the need for computer controlled opponents to show pseudo-intelligent behaviours. From an entertainment point of view, there is no need for this behaviour to be comparable to human intelligence, yet it should be intelligent enough to entertain the person that is playing the game. A classic example of an entertaining game AI can be seen in the game Pac-Man. This game implements a basic form of AI where each ghost moves, based on a simple set of rules, through the game environment with an increasing speed. With the growing realism and high fidelity in modern computer games, players expect much more from the game AI. AI controlled NPCs are expected to patrol in formations, exhibit squad based tactics, call for reinforcements, take cover from fire and retreat when facing a losing battle [80].

Indeed, the benchmark of "standard" game AI is rising, yet its growth is greatly outpaced by other components of gaming such as special effects animation, game mechanics design and in-game kinematics modeling. Game AI technology has been performing poorly for the following reasons. First, modern games tend to be very complex, featuring many different interacting objects, incomplete information, noisy environment and a large variety of possible actions at any given game instance. Second, there are severe time constraints on game AI to make real time decisions [27] [28]. It must be capable of solving real time decision task quickly, rationally and satisfactorily in a dynamic adversarial environment [100].

In general, academic research in AI centres around the development of automated inference machines and algorithms that infer certain consequences or outcomes based on a certain set of existing conditions. The techniques designed to achieve this can roughly be categorized into two schools of thought, conventional AI and computational intelligence (CI). Conventional AI includes methods such as expert systems, case based reasoning, Bayesian networks and behaviour-based AI. These systems are usually characterized by formalism and statistical analysis and attempts to mimic human intelligence through knowledge bases. Deep Blue of 1997 can be considered as a classical demonstration of conventional AI.

Computational intelligence on the other hand is known for its use of informal, non-statistical and often trial and error approaches. Learning, in its case, is an iterative process based on empirical data and is often associated with soft computing. Techniques such as neural networks, fuzzy systems, swarm intelligence and evolutionary computation fall under this classification. The branch of computational intelligence adopts a philosophical belief that intelligence is often too complex and computationally intractable to solve by the clear, elegant and homogenous systems as advocated by conventional AI methods.

This does not mean that these two approaches to AI are mutually exclusive. Existing research have established the viability and capability of using CI techniques to complement conventional AI. In addition, domain knowledge can be presented to guide the training process in achieving fast, accurate and efficient learning. CI techniques automate the process of finding a good solution, without the need to undergo the tedious cycle of devising the scheme of problem solving through manual means. This not only lowers the efforts expended remarkably but also adds value by increasing the potential of deriving solutions that are better than using either approach alone. This thesis proposes methods of developing techniques from computational intelligence, some inspired by ideas from conventional AI, with the focus on enhancing the player experience in computer games.

1.2 Types of computer games

In mainstream media, computer games are often categorized into many genres such as first person shooters (FPS), real time strategy (RTS), role playing game (RPG), adventure, simulation, etc. And many more hybrid genres exists such as action-adventure, role playing strategy, and more are being created as the industry develops. The point to note from this is that computer games are grouped according to the underlying game mechanics and the types of skills required to play the game. Such classifications are not so useful from a research standpoint. Instead, the three categories of computer games put forward by Togelius will be discussed [175]: computerized games, management games, and agent games.

Computerized games are games that tend to have discrete state spaces and a clear set of rules. Games in this category include board games such as Chess and Checkers, card games such as Poker and Bridge, and puzzle games such as Sudoku and Picross. These games generally do not require high amounts of computational resources to implement and a majority of them can be played without using a computer at all. The simplicity of implementing such games makes them a convenient benchmark for comparing the performance different AI algorithms, as well as between and against human players. However, the nature of these games also makes them unsuitable for investigating human cognition and perception.

Management games are games where the player takes a more macro role in the game world. These games often involve some form of economic, warfare, or life simulation. In these games, the player does not control any single character in the game but instead devises strategies, allocates resources, sets goals, and schedules productions in order to advance the game. Games in this category include real time strategy games such as Warcraft and Starcraft, god games such as The Sims, sports management games such as Championship Manager, and civilization games such as Civilization. These games tend to be complex, featuring multiple interconnected game mechanics, incomplete information and noisy environments. As with computerized games, management games are usually unsuited for research into cognition and perception issues.

Agent games are games where the player directly controls a character or agent within a game environment. The player decides where the agent goes and what the agent does at all time during the game. Games in this category include platform games such as Super Mario Bros and Rayman, arcade games such as Pac Man and Space Invaders, racing games such as Need for Speed and Gran Turismo, fighting games such as Street Fighter, and action games such as Grand Theft Auto. Agent games are well suited for investigating cognition and perception because the agent that is being controlled by the human player in the game environment is said to be both situated and embodied. That is, the agent is represented by a body in the game environment and is able to interact, affect, and perceive the world and its body through its actions. These games tend to play out in real time, hence placing additional constraints on the performance of its AI. This thesis investigates the issues of enhancing player experience through the use of agent games. In particular, chapter 4 of this thesis proposes and describes in detail a framework for a computationally efficient game AI suitable for implementation in real time games. The framework is generic enough to be applied to any agent games where the game AI can be expressed as a combination of behaviours. The proposed framework is tested using a real time car racing simulator game. The resulting car driver is able to outperform previously unseen opponents in direct competition, and is also the most computationally efficient.

1.3 Player experience

The most prominent role of computer games is one as a form of entertainment. Therefore, it is important for game developers to produce games that are entertaining, satisfying, and fun. Game designer Raph Koster said that for a game to be fun, the level of challenge need to be approximately right [79]. A game that is too easy or too difficult is perceived as boring. In a similar way, Thomas Malone described the essence of fun in three categories: challenge, fantasy and curiosity. In challenge, there needs to be a goal in the game to provide entertainment value but this goal should not be too easy or too hard to achieve [91]. 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 [38]. An expert player may be bored by a weak computer controller opponent while the same opponent may pose too much difficulty to a novice player. Hence, adaptability is an important consideration in a game AI. The core game AI that is encoded in a game needs to cater to a wide variety of audiences who play the game. In addition, these players learn to play the game better over time, so the game AI needs to scale appropriately to continually provide sufficient challenge to the player. Furthermore, such an adaptive game AI implementation will have the potential to customize a personalized and entertaining game experience to a specific player. Chapter 5 of this thesis 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 algorithm parameters are investigated for both algorithms and a general rule of thumb for the selection of these two parameters is proposed. The key contribution of this algorithm is the absence of a training phase. This way, the human player can immediately feel the effects of adaptation without having to play several games first just to train the game AI.

A believable game AI can help players to immerse in the game world, thereby making the game more enjoyable and satisfying. Murray defines immersion as a metaphorical term to describe the sensation of being surrounded by a completely other reality [99]. Believability in a game is one way of achieving such an immersion and maintains the suspension of the player's disbelief. The concept of suspension of disbelief was first coined by Coleridge in 1817 to describe the quality of a good fiction to make readers accept the unexplained or seemingly irrational aspects of the story for the purpose of enjoying the story. Extending this concept to the context of computer games, a believable game agent is one whose actions appear lifelike, rational, and allows the player to suspend disbelief [93]. Bryant also argued that an intelligent game agent must sometimes go beyond the ability to complete a task by completing it in a visibly intelligent manner [24]. Chapter 6 of this thesis focuses on evolving believable movement behaviours in game agents using two ideas, namely, introducing sensor noise to simulate errors in human judgment, and using action histograms to indirectly model idiosyncrasies in human controlled game agents. Game agents are evolved using a multi-objective approach to optimize the incomparable objectives of performance and believability. In a user study involving 58 respondents, the proposed game agents are found to be more believable compared to one optimized for performance alone.

1.4 Contributions

This thesis describes in detail a number experiments and studies, many of which form the premise for subsequent ones, that explore the primary aim of investigating and developing novel computational intelligence approaches to enhance the player experience in real time computer games. This section will summarize the main achievements and contributions of this thesis to advance the state-of-the-art of AI in computer games.

- A framework for designing a computationally efficient agent game AI based on a hybrid evolutionary behaviour-based methodology is introduced. This method is shown to have successfully and automatically exploited some collaboration between the different behaviour components which may have gone unnoticed if designed by hand. It is also easy for designers to incorporate symbolic domain knowledge without specifying its related parameters.
- A dynamic difficulty scaling and online adaptation algorithm is designed over the framework to increase player satisfaction. It has the advantage of being easily scalable by adding new behaviour components. The proposed adaptive algorithm learns during the game session and no offline training is required. This will allow new players to immediately feel the effects of the adaptive game AI. Newly introduced parameters are thoroughly investigated and a general rule of thumb for their selection is put forward.
- The action histograms and action sequence histograms are introduced as a means to analyze differences between game players (humans and AI). A case study is conducted to quantify the unnatural behaviours

seen in existing AI agents. The proposed histograms are shown to be successfully used as fitness functions to imitate low level behavioural tendencies of human players. The novel use of small window sizes of action sequences differs from conventional state-action approaches.

- Our experiments have introduced and verified the use of deliberate evolvable sensor noise in game AI agents to simulate systematic errors and random errors in human judgment during game playing. The introduction and co-evolution of these noise parameters is also demonstrated to improve the believability of AI agents.
- The believability of AI agents is shown to have the potential to be improved without degrading its game competency. A user study is conducted and the game AI agent evolved using the proposed histograms and sensor noise is verified as being more believable by human observers.

1.5 Thesis outline

This thesis is organized into seven chapters. The current chapter provides an introduction to computer games, game AI, and player experience, and motivates the research documented in this thesis. The primary aim of this thesis is to present an investigation on a computational intelligence approach to enhancing player experience in computer games. Two key issues of game AI affecting the player experience, adaptability and believability, are considered in this thesis.

Chapter 2 expands on the topic of computational intelligence and focuses on the main techniques used in this thesis. In particular, the basic framework of evolutionary algorithms, genetics algorithms, evolution strategies, co-evolution, multi-objective algorithms including Pareto dominance and optimality, and neural networks are discussed in this chapter.

Chapter 3 presents the real time car racing simulator game used in this thesis. The mechanisms for waypoint generation, vehicular controls, sensors model, and physics model are described in detail. Finally, the performance and characteristics of several heuristic controllers which were used as trainers in later chapters are discussed.

Chapter 4 proposes and describes in detail a framework for a computationally efficient game AI that is suitable for implementation in real time games. This approach combines the good response time of behaviour-based systems and the search capabilities of evolutionary algorithms. The proposed framework is demonstrated using the real time car racing simulator game and the evolved behaviours are quantitatively and qualitatively analyzed. The resulting car driver is then tested against previously unseen real world opponents written by other researchers.

Chapter 5 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 during the game itself. The objective of the adaptive algorithm is to match the game difficult to the proficiency of the game player to provide a suitable amount of challenge. Two indicators are also proposed as a measure of how well an adaptive algorithm is able to match its opponent.

Chapter 6 focuses on evolving believable game agents to improve the player's immersion in the game. Two ideas, namely sensor noise and action histograms, are introduced to induce believable movement behaviours in the game AI. A multi-objective approach is applied to simultaneously optimize both game performance and believability in the game agent. A user study is also conducted to quantify the improvement in believability achieved by this approach.

Finally, a high level summary of this thesis and some directions for future work are discussed in chapter 7.

Chapter Two

2 Computational intelligence

Computational intelligence is part of the larger family of computer science and engineering. The field of computational intelligence encompasses techniques such as artificial neural networks, evolutionary computation, fuzzy logic systems, ant colony optimization, particle swarm optimization, and artificial immune systems, etc. The computational intelligence approaches that are used in this thesis will be introduced in this chapter.

2.1 Elements of evolutionary algorithms

Evolutionary algorithms are stochastic, population based search algorithms that are inspired by Darwin's theory of evolution. It implements several evolutionary approaches found in nature such as selection, reproduction, crossover and mutation, amongst others, to improve the survival chances of a population over several generations. It follows the basic principle of survival of the fittest. Each element in the evolutionary algorithm framework will be discussed in this section.

2.1.1 Overview

In nature, all organisms have their unique set of genes. During the reproduction process, these genes are recombined by the process of gene crossover to form an offspring that carries characteristics from both parents and occasionally new characteristics by gene mutation that may or may not be beneficial. All organisms are then tested in their environment and only the ones most suited for the environment will survive to propagate their genes to the next generations.

Evolutionary algorithm uses these elements in an algorithm to solve complex optimization problems via a population of candidates. Each individual in the population consists of a set of variables that forms the solution to the problem. Individuals are tested and sorted according to their performance and those that perform better are more likely to be selected as parents to reproduce. The selected individuals exchange information by merging or swapping parts of their solutions to form a new population of offspring. The cycle then repeats itself by testing and sorting the new population of candidates. After substantial iterations, the algorithm should evolve a solution that is optimal for the problem. This process can be better visualized in the form of a flowchart of a basic genetic algorithm shown in Figure 2.1.

Other than nature inspired genetic operators such as crossover and mutation, computer scientists have also introduced new mechanisms which are not found in nature to evolutionary algorithms. An example of such is the concept of elitism. Some of the fittest individuals in the current population are cloned to the next generation without modifications so as to ensure that good solutions found in this generation will not be lost through the recombination operators. Such mechanisms can improve the performance of evolutionary algorithms over the course of the search process.

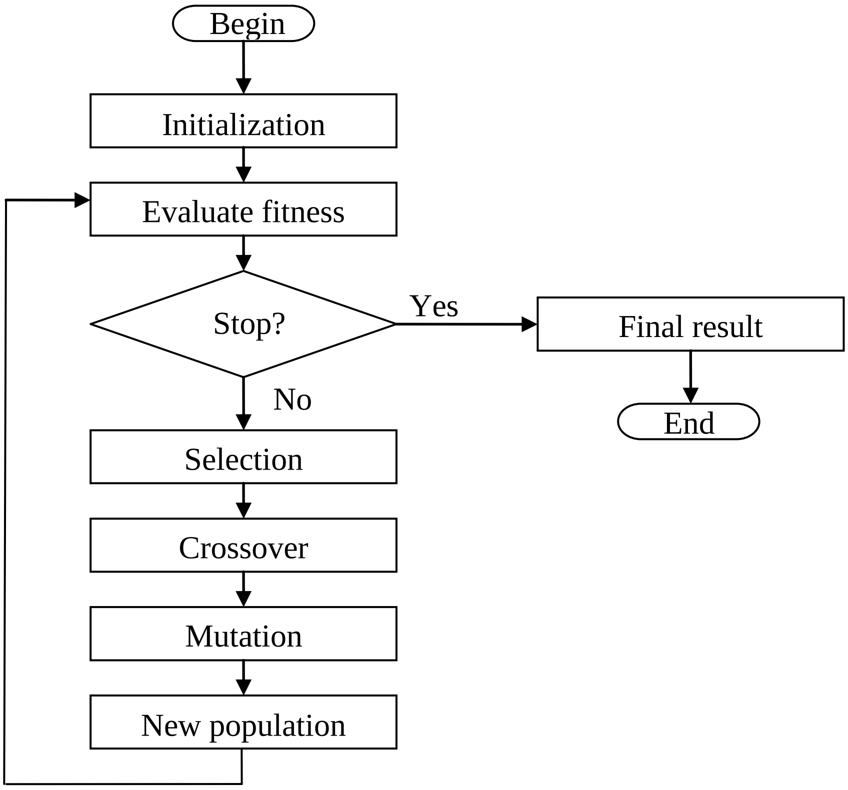


Figure 2.1 Flowchart of genetic algorithm

2.1.2 Representation

Just as genetic information is encoded in the DNA of living organisms, the solution to the problem in an evolutionary algorithm is encoded in the chromosome of an individual. In other words, each individual in the population encodes a solution to the problem. The manner in which a solution is encoded in an individual is referred to as the representation. For example, the integer value 8 can be represented simply as an integer variable ‘8’ or it can be represented as a binary value ‘111’. The representation directly affects the performance of the evolution. If a chosen representation is not generic enough to cover the entire search space, then such regions will become inaccessible to the evolutionary algorithm and good solutions within such regions will not be found. For example, an individual represented by integer variables will be unable to optimize a problem where the optimal parameters are real numbers. Therefore, it is important to design representations that are well suited for the problem. Some popularly used representations include real number, binary or more complex data structures such as tree nodes and neural network nodes.

2.1.3 Fitness and evaluation

The fitness of an individual is the criteria by which the environment evaluates the individual. An individual of high fitness is said to be well suited for the environment and will likely survive on to subsequent generations. In nature, the typical measure of fitness is the lifespan of an organism. The longer an organism is able to survive, the more opportunities it will have to reproduce and create offsprings. In evolutionary algorithm, the fitness of an individual is measured by the goodness of the solution it represents. For example, in function maximization problems, the fitness is simply the function output; the higher the function output, the fitter the individual. The fitness value is then used to determine the extent to which an individual is allowed to reproduce for the next generation.

2.1.4 Population and generation

Evolutionary algorithms use a population-based approach in its search process. A population consists of a predefined number of individuals which will evaluate different parts of the search space. In the beginning, the individuals in the population are randomly initialized to populate the search space. Each individual in the population will be evaluated to determine its fitness. When all the individuals in a population have been evaluated, recombination will be performed and a new population of offspring will be created. With the creation of the population of offspring, one generation or one evolutionary cycle is said to have elapsed. A large population size will typically survey a larger search space and increase the probability of finding good solutions at the expense of longer computation time. Depending on the complexity and difficulty of the problem, evolutionary algorithms typically require tens to thousands of generations before a reasonably good solution can be found.

2.1.5 Selection

Inspired by the laws of nature, a fitter individual in a population should be given a higher likelihood of survival and more opportunities to reproduce than a weaker individual. Nevertheless, the weaker individual should still be given some small finite chance of survival and propagation. Such a mechanism is realized in evolutionary algorithms as the selection process. In a popular implementation of the selection process known as the roulette wheel selection [8], each individual is assigned a probability of being selected based on its normalized fitness against the total population fitness. Hence, an individual with high fitness will have a higher probability of being selected for propagation while an individual with low fitness will still have a small but finite probability of being selected. A good selection mechanism should seek to maintain a balance of good and weak individuals in a population. Too high an emphasis on retaining good individuals may result in premature convergence and having the population trapped in local optima. Conversely, a high emphasis on retaining weak individuals may lead to low selection pressure and slow rate of convergence. A balance of exploration and exploitation is required for the good performance of evolutionary algorithms.

Other commonly used selection mechanisms include tournament selection [96] and rank-based selection [8].

2.1.6 Crossover

Crossover, or sometimes known as recombination, is the process where genetic information from two parent individuals are exchanged to produce an offspring. Such an offspring will receive characteristics from both parents in the hope that the new combination of genes will produce an individual that is fitter than both its parents. The crossover process is associated with a probability of crossover which determines the likelihood of a crossover taking place. The probability of crossover is typically set high so as to facilitate the exchange of search information between individuals and improve the efficiency of the algorithm. The actual implementation of a crossover operation is often problem and representation dependent. Some commonly used crossover mechanisms [59] include single-point, multi-point, uniform, shuffle, arithmetic, and order based crossovers.

2.1.7 Mutation

Mutation denotes the random modification of some genetic material of an individual. Although mutations are often viewed as being harmful, they may also be beneficial in some instances and may result in individuals that are more fit when compared to it predecessors. In evolutionary algorithms, mutation is necessary to preserve diversity in a population. That is, mutation helps to maintain the exploration ability of the population and to escape from local optima should the population become trapped. As with the crossover operation, the mutation operation is associated with a probability of mutation which determines the likelihood of a mutation taking place. When used in conjunction with the crossover operation, the probability of mutation is typically set low so as to maintain diversity in the population without disrupting the flow of the population. In the absence of the crossover operation, the probability of mutation is set high as it becomes the main mechanism for exploration. The actual implementation of a mutation operation is often problem and representation dependent. Some commonly used mutation mechanisms include bit-flip mutation, position swap, and Gaussian perturbation.

2.1.8 Elitism

Elitism is an example of a process not found in nature but was introduced to evolutionary algorithms to improve its performance. It was first conceptualized by De Jong [39] to preserve the best individuals found and prevent the lost of good solutions due to the stochastic nature of evolutionary processes. It is implemented in evolutionary algorithms by simply copying the fittest individuals in the population to the next generation without any alterations. Elitism ensures that the minimum fitness of a population never decreases across generations and typically results in a higher rate of convergence. In practice, the implementation of elitism requires the algorithm designer to specify a percentage of individuals from the parent population to directly replace the same percentage of the weakest individuals in the offspring population.

2.1.9 Stopping criteria

The stopping criteria refer to the conditions which will stop the evolutionary algorithm when met. This is an important consideration as both computational resources and time are limited and it is not practical to allow an algorithm to run indefinitely. A good stopping criterion will allow sufficient resources for the evolutionary algorithm to convergence to good, if not optimal, solutions. Some commonly used stopping criteria include setting a desired fitness level, setting a maximum number of generations, stopping when the fitness level stagnate for some number of generations, and stopping when the standard deviation of fitness level stagnate.

2.2 Genetic algorithms

Genetic algorithm (GA) [67] was introduced by Holland in the 1970s. The basic GA consists of a fixed population size, a fixed length of chromosome represented by binary strings, and uses a conventional objective function. It is typically applied to discrete optimization problems such as combinatorial problems. It emphasizes the use of crossover operators to combine information from good parents. The crossover and mutation operators work by flipping and swapping binary bits. The basic GA represents the general framework of evolutionary algorithms and many variants can be created by using the basic GA framework as a starting point. GA has been applied successfully to a wide variety of problems. An example from the finance industry would be futures trading [103]. The simplicity and flexibility of GA also makes it easy to hybridize with other computational intelligence techniques such as neural networks and fuzzy logic [75] [77], and also heuristics methods [143].

2.3 Evolution strategies

Evolution strategies (ES) [124] was introduced by Ingo Rechenberg and Hans-Paul Schwefel in the 1970s. ES is particularly suited for real valued optimization problems because its solutions are represented as real numbers. It uses only the mutation operator and does not use any crossover operators. A special feature of ES is the inclusion of self-adapting mutation parameters as a standard procedure in its algorithm. The self-adapting mutation parameters are encoded together with the solution in the chromosome hence making the chromosome twice as long. ES also defines two types of selection mechanisms, namely plus and comma strategy. In the plus strategy, μ parents participate in the production of λ offsprings. Next, the λ least fit individuals are removed from the $\mu+\lambda$ individuals and the remaining individuals form the new generation. The plus strategy always retains the best solution and can get stuck in local optima. In the comma strategy, μ parents participate in the production of λ offsprings, but the new generation will be selected from λ offspring individuals only. The advantage of this is that the comma strategy is better in escaping from local optima but the disadvantage is that it might lose the individual with the best solution.

2.4 Co-evolution

Co-evolution can be classified into two main classes, namely competitive co-evolution and cooperative co-evolution. In this section, only competitive co-evolution will be describe as it is the only paradigm that will be used in the experiments. A good read on the topic of co-evolution can be found in [35].

The competitive co-evolution model [52] [64] [130] is often described as a host-parasite or predator-prey interaction. This is implemented as two sub-populations in an evolutionary algorithm. One sub-population represents the potential solution to the problem while the other sub-population acts as fitness tests. Each sub-population will evolve and adapt to counter the other sub-population in order to become the new winning sub-population. This results in an evolutionary arms race as each sub-population tries to exploit weaknesses and outperform the other sub-population. This has the advantage of ensuring that neither sub-population becomes over trained and thereby losing generalization capability. The resulting solution will likely be good and generic. Successful applications of co-evolution can be found in pursuit evade games [105], multi-agent games [201], strategy games [7] and board games [72] [87] [131].

The actual implementation of co-evolution in evolutionary algorithms is problem dependent. Hence, co-evolution is better viewed as an implementation concept more than a specific technique. In the experiments, co-evolution is implemented by playing members of the same population against each other in a two player game. Individuals within the same population will exploit weaknesses of other individuals. The resulting population is one that is continually changing and rooting out weak traits from the population.

2.5 Multi-objective optimization

Some types of problems involve multiple objectives that are competing and incomparable. Such problems are said to be multi-objective (MO) problems. For example, consider the problem of commuting from home to the university everyday. Two possible objectives that one may take into consideration are the cost of transportation and the time required for the journey. The cheapest form of transportation may be the public bus but it is also the slowest. The fastest form of transportation may be to take a taxi but that is also the most expensive.

Generally, many real-world applications involve complex optimization problems with various competing specifications and constraints. A minimization problem with decision space, X, a subset of real numbers, can be used without loss of generality. For minimization problems, it tends to find a parameter set P shown in (2.1).

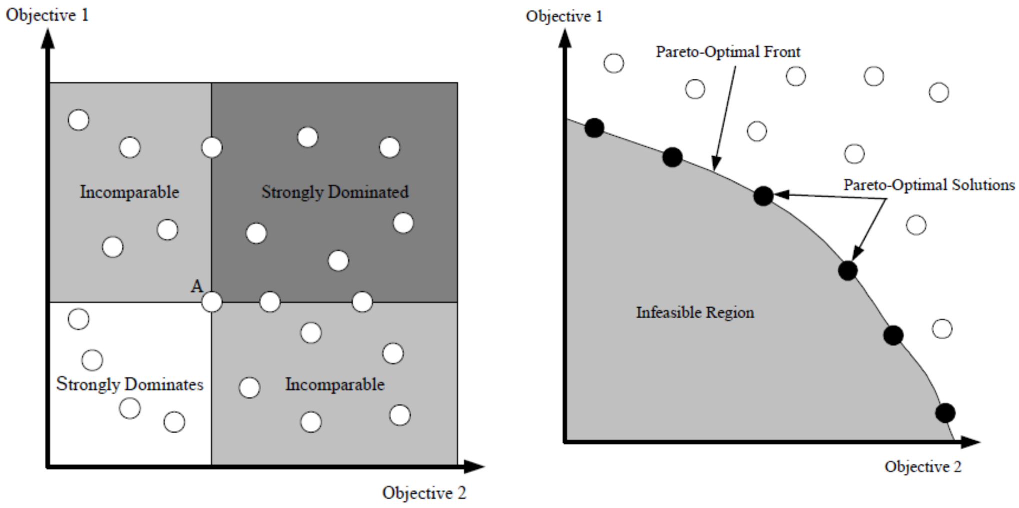
$$\underset{\boldsymbol{P}\in X}{\text{Min}}\, \boldsymbol{F}(\boldsymbol{P}),\ \mathbf{P}\in\mathbf{R}^{D} \tag{2.1}$$

where $P=\{p_1, p_2, ..., p_D\}$ is a vector with D decision variables and $F=\{f_1, f_2, ..., f_M\}$ are M objectives to be minimized.

The solution to MO optimization problems exist in the form of an alternate tradeoff known as Pareto optimal set. A single objective component belonging to any non-dominated solution in the Pareto optimal set can only be improved at the expense of degrading at least one of its other objective components. A vector F_a is said to dominate another vector F_b, denoted as shown in (2.2).

$$F_a \prec F_b, iff\ f_{a,i} \le f_{b,i} \forall i=\{1,2,...,M\}\ and\ \exists\, j\in\{1,2,...,M\}\ where\ f_{a,i} \prec f_{b,i} \tag{2.2}$$

In the absence of specific domain information regarding the preference of objectives, a ranking scheme based on Pareto optimality is regarded as an appropriate approach to representing the fitness of an individual in evolutionary MO optimization problems [48]. The concepts of the Pareto dominance relationship and the Pareto-optimal front are illustrated in Figure 2.2.



(a) (b)

Figure 2.2 Illustrations of (a) Pareto dominance relationship and (b) Pareto-optimal front

The solution to MO optimization problems exist in the form of an alternate tradeoff known as Pareto optimal set. A single objective component belonging to any non-dominated solution in the Pareto optimal set can only be improved at the expense of degrading at least one of its other objective components. Each solution in the population is given a Pareto rank given by equation (2.3).

$$rank(i) = 1 + n_i \tag{2.3}$$

where n_i is the number of solutions in the population dominating the individual i in the objective domain. In a Pareto optimal set, each solution is fitter than any other solution in at least one objective. The solutions with a lower Pareto rank have a higher likelihood of being selected as parents for the next generation. At the end of the algorithm, a decision is made to determine the most suitable solution for the intended problem. Examples of MO algorithms include non-dominated sorting genetic algorithm II (NSGA-II) [40], strength Pareto evolutionary algorithm 2 (SPEA2) [202] and Pareto archived evolution strategy (PAES) [78] etc. A good introduction to multi-objective optimization can be found in [36].

2.6 Neural networks

Artificial neural networks (or simply neural networks) [63] are a class of machines that are designed to model the way in which the brain performs tasks. In particular, these machines must exhibit the behaviour of learning and the ability to store knowledge. A neural network consists of an interconnected group of artificial neurons designed to model some properties of biological neural networks.

Neural networks have been widely used for image processing, speech processing, pattern recognition [90] [162] [185], function approximation [83], and time series prediction. Its applications can be seen in many areas like control problems [55], medical diagnosis, robotics [71], game AI [126] [139], financial analysis [192], criminal investigation, and even driving a car [13] [15]. The main reason for the successful application of neural networks is its ability to learn and generalize well to unseen situations.

2.6.1 Multi-layer perceptrons

Neural networks consist of input units, hidden units and output units call nodes. Each node (or perceptron) has an activation function, which acts as a mapping function. Each connection has a strength represented by a weight, which help to define the input-output relationship of the network. A simplified view of a multi-layer perceptron (MLP) is shown in Figure 2.3.

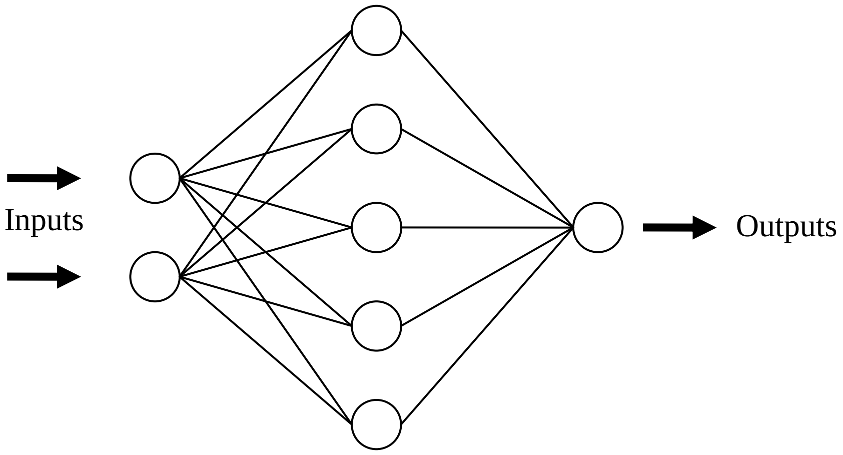


Figure 2.3 A simplified view of a MLP

Neural networks can be trained using a paradigm known as supervised learning. In supervised learning, a set of example and their desired outputs, known as a training set, is available from experiments. The examples from the training set are shown repeatedly to the neural network and an output is produced from the neural network. If this output is different from the desired output, then the neural network adjusts its weights to make an improvement. This process is known as the training algorithm. The aim of the training algorithm is to minimize the error function as shown in (2.4)

$$J(W) = \frac{1}{2}\sum_{j=1}^{N}\left(d(j) - y(j)\right)^2 \tag{2.4}$$

where W is the weight vector of the neural network, d(j) is the desired output of the j-th example and y(j) is the neural network output of the j-th example.

The error function is to be minimized using the right values for the weights W via the gradient descent method. The computation of the gradient can be obtained using a method called the backpropagation which efficiently exploits the use of the chain rule on composite functions. For conciseness, the detailed workings of the backpropagation algorithm can be found in textbooks [63] and shall not be discussed here. One of the main disadvantages of the gradient descent method is its long computation time. Methods to find the global optimum are usually very computational expensive if not impossible at all.

2.6.2 Evolutionary neural networks

As the name implies, evolutionary neural networks are the hybrid between evolutionary algorithms and neural networks. The training of neural networks involves finding a set of weights that will generate the desired output for a given input. As discussed in the previous section, such weights can be optimized by the backpropagation algorithm. Alternatively, one may also use evolutionary algorithms, which is itself an optimization technique, to optimize the weights of the neural network. A good introduction to the field of evolutionary neural networks can be found in [200].

In evolutionary neural networks, the fitness function is often defined as the minimum sum of square errors between the outputs of the neural network and the desired outputs from the training data. In its simplest form, the number of hidden units in the neural network is predefined by the user and the weights of the neural network are represented as an array of real numbers in the chromosome of the evolutionary algorithm. However, many variants are also possible, such as automatically evolving the number of hidden units in the neural network [54], constructing recurrent neural networks [6] and evolving the entire topology of the neural network [149] [150] [151].

2.7 Summary

In this chapter, the computational intelligence approaches that are used in this thesis are introduced. The core elements of the evolutionary algorithm framework are discussed in detail as a good understanding of these basic building blocks will allow us to improve their performance and find new applications for them. Two specific implementations, namely genetic algorithms and evolution strategies, are primarily used in the experiments. Other concepts such as co-evolution and multi-objective optimization are used in conjunction with the basic framework to allow evolutionary algorithms to be applied to a wider range of problems. This is followed by the introduction of artificial neural networks and the use of evolutionary algorithms as their training methods.

Chapter Three

3 Real time car racing simulator

The car racing simulator model used in the experiments is modified from the one used in Simulated Car Racing Competition held during IEEE CEC 2007 [179] [180]. The main features of the simulator will be summarized in this section. In this game, up to 2 players drive their cars in an open arena and earn points by driving through an ordered sequence of waypoints. In a 2 player game, the player with more points at the end of the stipulated game time wins the game. In a 1 player game, it becomes a reverse time trial as the player tries to achieve as high a score as possible within the stipulated game time. An illustration of the game arena in a 2 player game is shown in Figure 3.1.

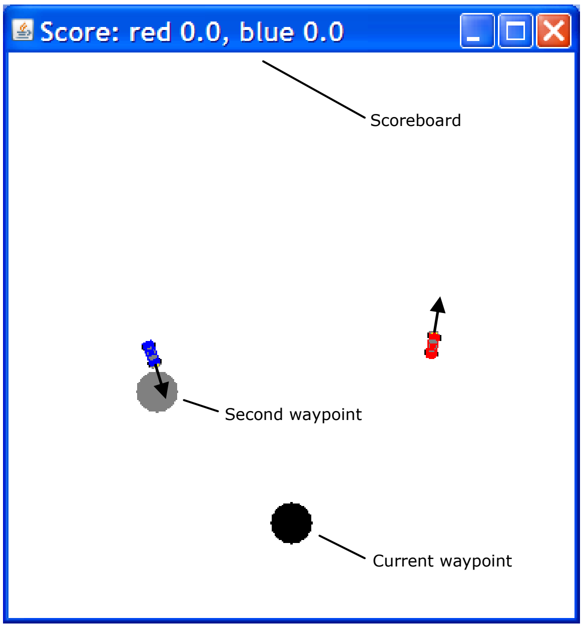

Score: red 0.0, blue 0.0

Figure 3.1 The real time car racing simulator game area

3.1 Introduction

The time keeping system in games can be broadly categorized into two types, namely turn based and real time. In the turn based system, players in the game take turns to perform actions within the game. That is, while one player is performing his actions, the other players may only observe the game environment without any active participation. Only when the current player completes his turn may the next player begin his turn. Conversely, in a real time system, game time passes continuously according to a global game clock. All players perform their actions simultaneously and at the same time observe the effects of their opponents' actions and response in real time. Hence, there is an element of time management involved in real time games. In real world games, chess is a classic example of turn based games, while ball sports such as basketball and soccer and examples of real time games.

The dimension of the competition field is 400 pixels by 400 pixels and is not occupied by any walls or obstacles. As such, vehicles are free to drive outside of 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 their current position is outside the game field. Assuming the lower left corner to be the origin, the starting position of the first vehicle is fixed at the coordinate (100, 200) while the starting position of the second vehicle is fixed at the coordinate (300, 200).

The objective of each race is to drive through 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 can be driven through but is not worth any points unless the current waypoint is driven through first. That is, the waypoints must be driven through in an ordered sequence. Nothing will happen when a car drives through the next waypoint. Whenever the current waypoint is driven through, the car that drove through it gains 1 point, the next waypoint will become the new current waypoint and a new waypoint will be generated to replace the next.

It should also be noted that in the context of computer games, real time games are not subjected to real time constraints such as operational deadlines from event to system responses encountered in real time control systems. Computer games are typically produced to operate at 30 frames per second. In the event that the game AI requires more than 1/30 seconds to response, the frame will be dropped. The result is a game that runs at a lower frame rate. However, it is still desirable for game AI to be computationally efficient so that frame rates can be maintained at 30 fps or more to provide a better playing experience.

3.2 Waypoint generation

There are several ways to generate the new waypoint. One such method is to randomly generate a new waypoint anywhere within the playing area. 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 will always be generated on the circumference of a circle of radius (400/3) pixels centred on the k-th waypoint within the visible competition area. The arbitrarily chosen radius is to ensure that 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 centre of the field. This is to ensure that the initial conditions are not in favor of any vehicle in particular.

Another way to generate the waypoints is to make use of a stored array of waypoint coordinates. That is, a sufficiently long list of waypoints, usually about 35 waypoints for a game of 1000 time steps, is generated before the game begins. Whenever a waypoint is passed, instead of generating a new waypoint on the fly, it is simply loaded from the next item on the list. This allows the designer to predefine a unique route for the player to drive through. This can be used to simulate a virtual race track where the designer can incorporate difficult situations such as creating routes that require very small turning radii. Such designer tracks can also be used in control experiments to benchmark the performance of the human or AI players.

Accelerate and Steer left 6	Accelerate 7	Accelerate and Steer right 8
Steer left 3	Neutral 4	Steer right 5
Decelerate / Reverse and Steer left 0	Decelerate / Reverse 1	Decelerate / Reverse and Steer left 2

Figure 3.2 Graphical representation of the controller and its corresponding integer value in the Java Controller interface

3.3 Vehicle controls

The vehicles themselves are controlled using digital trigger type controllers like the directional controls found on console game control pads. The 4 distinct on-off control signals: accelerate (up), decelerate (down), left and right turn combine to form a total of nine possible controller states, inclusive of a neutral state where no key is asserted. This is better illustrated in Figure 3.2.

The controller can take inputs either from the keyboard or an AI algorithm. On a keyboard, accelerate and decelerate actions are mapped to the up and down arrow keys respectively, while the left and right actions are mapped to the left and right arrow keys respectively. If no keys are depressed, then a neutral action is asserted. In the AI controller, the control mechanism of the car in the racing game is implemented via the Java Controller interface that returns an integer value from 0 to 8, which represents the nine possible controller states, to the game engine at each time step.

Table 3.1 Full list of sensors available in the real time car racing simulator

Sensor name	Description
getSpeed	Double. Returns the speed of the controlled vehicle.
getAngleToNextWaypoint	Double. Returns the angle of the currently activated waypoint from the controlled vehicle in radians, -π to π.
getDistanceToNextWaypoint	Double. Returns the distance of the currently activated waypoint from the controlled vehicle.
getAngleToNextNextWaypoint	Double. Returns the angle of the next activated waypoint from the controlled vehicle in radians, -π to π.
getDistanceToNextNextWaypoint	Double. Returns the distance of the next activated waypoint from the controlled vehicle.
getAngleToOtherVehicle	Double. Returns the angle of the other vehicle from the controlled vehicle in radians, -π to π.
getDistanceToOtherVehicle	Double. Returns the distance of the other vehicle from the controlled vehicle.
otherVehicleIsPresent	Boolean. Returns true if the other vehicle is present (i.e. 2 player game) or false otherwise (i.e. 1 player game).
justPassedWaypoint	Boolean. Returns true at the time step that a waypoint is passed by either vehicle or false otherwise.
otherVehicleJustPassedWaypoint	Boolean. Returns true at the time step that a waypoint is pass by the other vehicle or false otherwise.
getPosition	Vector. Returns the x and y coordinates of the controlled vehicle.
getVelocity	Vector. Returns the x and y component of the velocity of the controlled vehicle.
getOrientation	Double. Returns the direction in which the controlled vehicle is facing in radians, -π to π.
getAngularVelocity	Double. Returns the angular velocity of the controlled vehicle in radians per time step.
getDirectionOfMovement	Double. Returns the direction in which the controlled vehicle is traveling in radians, -π to π.
getOtherVehiclePosition	Vector. Returns the x and y coordinates of the other vehicle.
getOtherVehicleVelocity	Vector. Returns the x and y component of the velocity of the other vehicle.
getOtherVehicleOrientation	Double. Returns the direction in which the other vehicle is facing in radians, -π to π.
getNextWaypointPosition	Vector. Returns the x and y coordinates of the currently activated waypoint.
getNextNextWaypointPosition	Vector. Returns the x and y coordinates of the next activated waypoint.

3.4 Sensors model

The AI controllers can access but not modify the full state of the game in a third-person representation similar to that used internally by the game. Additionally, controllers can access much of the information in a more convenient first-person perspective, e.g. angles and distances from the frame of reference of the car. The full list of sensors available and their description is listed in Table 3.1.

3.5 Mechanics

In the simulation, a vehicle is specified by its position, velocity, orientation and angular velocity. The equations that govern these variables are given in equations (3.1) to (3.4).

$$s_{t+1} = s_t + v_t \tag{3.1}$$

$$v_{t+1} = v_t \times \left(1 - c_{drag}\right) + f_{driving} + f_{grip} \tag{3.2}$$

$$\theta_{t+1} = \theta_t + \dot{\theta} \tag{3.3}$$

$$\dot{\theta}_{t+1} = f_{traction} \times \left(f_{steering}() - \dot{\theta}_t \right) \tag{3.4}$$

where s_t is the position of the vehicle at time t, v_t is the velocity of the car at time t, c_{drag} is a scalar constant which is set to 0.1, $f_{driving}$ is the driving force provided by the vehicle engine which is set to 4 for acceleration, 2 for deceleration and 0 for neutral, f_{grip} is the force between the tires and the ground surface, its magnitude is set to 2 and its direction is set to 0 when the orientation of the vehicle and the direction of movement differ by less than $\pi/16$, $\theta - (\pi/2)$ when the difference is positive and $\theta + (\pi/2)$ when the difference is negative, θ_t is the orientation of the vehicle at time t, $\dot{\theta}_t$ is the angular velocity of the vehicle at time t, $f_{traction}$ limits the change in angular velocity to between -0.2 and 0.2, and $f_{steering}$ is the magnitude of v_t if the vehicle is steering left and negative magnitude of v_t if steering right.

In 2 player games, vehicle collisions are also modeled in the simulation. Collision is detected by checking whether the rectangular spaces occupied by the vehicles on screen intersect each other. When a collision is detected, the collision resolution methods on both vehicles are called. The velocities of both vehicles are then exchanged and both vehicles are shifted several pixels away from each other to undo the intersecting spaces in order to prevent repeated collisions in the next time step. Next, the angular velocities are updated by equation (3.5).

$$\dot{\theta} = \dot{\theta} \pm \frac{mag\left(v_{other}\right) + mag\left(v_{this}\right)}{2} \tag{3.5}$$

where mag() is the magnitude function of a vector, v_{other} is the velocity of the other vehicle, v_{this} is the velocity of this vehicle, and the sign of the operation depends on the relative position of the point of collision to the centre of the vehicle.

The included physics model is reasonably detailed, allowing for collisions between vehicles as well as side skidding. When cornering, a technically skilled controller, human or not, will be able to execute such maneuvers to their advantage.

When driving towards the current waypoint, care must be taken not to approach it at too high a speed. A driver that is accelerating and steering towards a waypoint may overshoot it and ends up orbiting around the waypoint. Additionally, if a driver overshoots the current waypoint, it may be put at a disadvantage if the next waypoint is positioned behind the car. Driving at a slower speed may help in most situations but runs the risk of losing the current waypoint to the opposing driver and therefore wasting valuable time driving towards a waypoint that it is unable to win.

Using a more aggressive approach, the driver may choose to intentionally collide with the opposing driver in the hope of throwing its opponent off course and thereby increasing its own chances of arriving at the current waypoint first. Alternatively, the driver may choose to throw itself in the path of its opponent to block its path and perhaps get a helpful bump towards the current waypoint. However, this requires that the driver itself be competent in recovering from collisions and also be able to predict the most likely outcome of a collision in order to determine whether or not it is advantageous to do so.

From a more tactical point of view, the driver can try to predict which driver will reach the current waypoint first. Given the situation, the driver can choose to drive faster towards the current waypoint or drive towards the next waypoint and wait there instead. This way, the driver loses a point for the current waypoint but wins the next point once it becomes activated and all is square for the race to the waypoint after that. However, predicting which driver will reach the current waypoint first requires a good understanding of the game dynamics as well as the driving behaviour of the opposing driver. Conversely, the opposing driver may also decide to forsake the current waypoint, in which case it becomes more logical to drive towards the current waypoint instead. In the example shown in Figure 3.1, both cars are roughly equidistance from the current waypoint. The red car is on the right and the blue car is on the left. However, the red car is facing away from the waypoint while the blue car is facing the waypoint directly. The red car can choose to reverse towards the waypoint to avoid wasting time to make a U-turn. But if the red controller knows that the reversing acceleration is slower than the forward acceleration, it will know that the race to the current waypoint is lost. In such a situation, the red car should forgo this waypoint and drive forwards toward the next waypoint instead. Additionally, the blue car, after driving through the current waypoint, will likely be back facing the next waypoint and hence be in a poor position to win the next point.

3.6 Example controllers

Three heuristic controllers with basic driving behaviours, which are packaged in the car racing simulator [179], will be described in the following sub-sections. These controllers are used in various experiments in this thesis. In particular, the HeuristicSensibleController is used as the main training opponent as it represents a well tuned naïve driver that places sufficient selection pressure on the evolving population. More sophisticated controllers have been developed [179], some by conventional AI and others by computational intelligence. However, the sophisticated controllers are not used during training because they encourage over training, specialized solutions and poor generalization. Therefore, only basic controllers are used during training to encourage better generalization.

3.6.1 GreedyController

The GreedyController (GC) is a simple controller that always outputs the acceleration motor command and there is no upper limit for its speed. It will steer towards the next waypoint intuitively depending on whether the angle to the current waypoint is negative or positive. This controller is rather ineffective in practice. When observed visually, it typically overshoots the waypoint due to its fast driving speed. In situations when the next waypoint is in the opposite direction, this controller needs to take a big detour, wasting valuable time.

3.6.2 HeuristicSensibleController

The HeuristicSensibleController (HSC) drives directly towards the current waypoint much like the GC but with an upper speed limit of 7 pixels per time step which is a moderate speed. If the instantaneous speed falls below the speed limit, it exerts the accelerate command but if it is above the limit it simply issues the neutral driving command. In general, this controller performs better than the GC in solo tests. The main drawback of this controller is that it does not have any waypoint prediction mechanisms. That is, it simply drives towards the current waypoint, disregarding whether or not it will reach the waypoint before its opponent does.

3.6.3 HeuristicCombinedController

The HeuristicCombinedController (HCC) is a more complex controller when compared to the HSC. Its behaviour will change depending on whether or not its present position is nearer to the current waypoint than its opponent. If it is nearer, it behaves identically to the HSC. However, if it is further away, the controller activates an “underdog” mode and drives towards the next waypoint instead, stopping in the vicinity of the next waypoint. In underdog mode, its speed limit is proportionate to the distance towards the next waypoint. It reduces to the HSC in solo races.

3.7 Summary

In this chapter, the real time car racing simulator used in this thesis is presented. Three methods of waypoint generation were discussed, the discrete control scheme used in the simulator was illustrated and the sensors available to the game AI were introduced. Next, the mechanics of the vehicles were presented along with some discussion on possible driving strategies. Finally, three simple heuristic controllers, which will be used as training opponents in the experiments, were described.

Chapter Four

4 Evolving computational efficient behaviour-based AI for real time games

This chapter examines the design of a game AI that is computationally efficient yet demonstrates highly competitive performance for a real time car racing simulator game. In turn based games, the game AI is able to compensate for its lack of game reasoning by evaluating board positions millions of times faster than the human player. However, such extreme resource requirements are impractical for fast paced and real time games, i.e. racing games, sports simulators, first person shooters and real time strategy games. This chapter proposes and describes in detail an evolved behaviour-based controller that combines the good response time of behaviour-based systems and search capability of evolutionary algorithms to evolve competitive driving behaviours for a real time car racing game. The proposed controller is tested against the top 5 participants in the Simulated Car Racing Competition held during the 2007 IEEE Congress on Evolutionary Computation to evaluate its generalization performance against previously unseen controllers. The proposed behaviour-based controller is able to outperform all its opponents in direct competition, and is also the most computationally efficient.

4.1 Introduction

The quality of commercial computer games is directly related to their entertainment value [182]. Game AI, being an essential part of a game, has become an important selling point of games [49]. In this aspect, game developers compete with one another by creating more sophisticated and intelligent game AI to offer better game play experiences. However, the current state of game AI is still, in general, of low quality [137]. There is a general dissatisfaction among game players with the level of the artificial intelligence of computer controller opponents. This has led to players preferring to play against human controlled opponents [137], via hot seat, split screen, local network, Bluetooth, infrared and most prominently the Internet. This group of players tends to value intelligent behaviours [157]. Such player preferences have also partly contributed to the boom in the development of massively multi-player online games in recent years. However, in situations where human game partners are unavailable, a competent game AI is still desirable.

World class game AI does exist and many examples have been developed that are able to beat good human players [89]. But these are generally restricted to slower paced, turn based, and perfect information games [27] such as Deep Blue for International Chess [74] and Chinook for Checkers [136]. Deep Blue compensates for its lack of game reasoning by evaluating individual board positions millions of times faster than the human player. These search methods can also be extended to multi-player games [68] [155]. However, such extreme resource requirements are impractical for commercial games where majority of the CPU time and memory is allocated to graphics rendering instead of AI. For faster paced, real time type of games, i.e. racing games, sports games, flight simulators, first person shooters (FPS) and real time strategy (RTS), such brute force evaluation methods are not feasible. In real time games, game time progresses continually and all players are required to perform their actions simultaneously. Computationally efficient methods that do not compromise in performance are necessary requirements for the implementation of game AI in such real time games [148].

Behaviour-based artificial intelligence (BBAI), which is popular in the field of robotics, provides some inspiration to address these real time computational requirements. In this methodology [22], intelligence is perceived as a large number of relatively simple and robust modular components. Each of these components work only within a specific set of conditions which it can identify from the environment. BBAI is reactive in nature and operates without search or deliberation, and is therefore very successful in time critical applications such robotics and interactive virtual reality [141] and suitable as game agents [69]. However, the disadvantage of reactive intelligence is in its design process since individual components need to be designed by hand.

Fortunately, computational intelligence (CI) techniques such as neural network, fuzzy logic and evolutionary computation have been demonstrated to be a valuable tool that can be employed to simplify the process of designing controller behaviours and to optimize its related parameters. Neural networks were trained as evaluation functions for checkers [32] [33] and as a targeting system for shooting games [58]. Evolutionary algorithms had been applied to evolve competent players and to analyze results in “Prisoner’s Dilemma” games [51] [119]. Hybrid fuzzy logic methodologies had been applied to incomplete information resource allocation games and network flow board games [21]. Genetic programming was applied to learn tactical behaviours by observing how human players perform in a driving simulator [46]. CI techniques have also been applied to physical cars in the DARPA Grand Challenge [167]. Game agent controllers employing CI techniques had been successfully applied to many games such as chess [43] [107], racing games [1] [30] [50] [65] [66] [112] [173] [174] [177], soccer simulation [101] [102] [113] [118] [133] [134] [154], role playing games [146], predator prey games [18] [41] [88] [193] [195], action games [37] [45] [108] [116] [186], puzzles [92] [97], real time strategy games [11] [23] [94] [114] [135] [153] [184], and even sumo wrestling [142] with reasonable performance against an average human player. CI techniques have also been used to design game contents [61] [62] [178]. While complex behaviours cannot be reliably and predictably evolved, simpler behaviours can be quickly found and thoroughly exploited. This characteristic suitably complements the process of designing the simple modules used in behaviour-based controllers.

This chapter examines the design of a computationally efficient controller for controlling a car in a real time car racing simulator game by using a hybridization of behaviour-based design and evolutionary computation search. The proposed controller will be referred to as behaviour-based controller for the remaining of this chapter. The behaviour-based controller will be evaluated and compared based on 2 metrics, computational efficiency and competitive performance. The best evolved controller will then be tested against the top 5 participants from the Simulated Car Racing Competition held during 2007 IEEE Congress on Evolutionary Computation (CEC) [180] in order to benchmark its generalization performance against previously unseen controllers.

The result of this design is a framework for a computationally efficient agent game AI based on a hybrid evolutionary behaviour-based methodology that is able to automatically exploit some collaboration between the different behaviour components which may have gone unnoticed if designed by hand. This demonstrates the possible synergy between conventional AI and computational intelligence.

4.2 Controller design

In this section, the design of the behaviour-based controller will be discussed in detail. First, an artificial neural network controller is evolved and its behaviour is analyzed. Learning from the relatively poor performance of the neural networks controller, a new set of component behaviours are proposed for implementation in the behaviour-based controller. The component behaviours are deliberately made generic and evolution strategies is employed to optimize the behaviour-based controller. A comparative analysis will be made between the neural network controller and the behaviour-based controller.

4.2.1 Neural network controller

An artificial neural network (ANN) [63] is a massively parallel distributed processor made up of simple processing units which has a natural propensity for storing experiential knowledge. A class of ANN employing multi-layer perceptrons (MLP) represents one of the widely used and effectively machine learning methods currently applied to data classification and function approximation problems. Although it usually takes a substantial amount of time to train a neural network, a trained network is computationally fast in its application due to it being a string of addition, multiplication and function mapping operations.

The neural network model is used to implement the game controller because a properly trained neural network can infer an output from a set of observational inputs. This is useful as it avoids the complicated task of analyzing the system and designing driving rules by hand. However, the disadvantage of this approach is that the resultant neural network acts as a black box control unit, making it difficult to make analysis or draw conclusions from the evolved weight values of the neural network. In this situation, the neural network controller is inspected and described visually to quantify its driving behaviours. In this section, a car racing controller constructed solely by neural networks is explored to study its potential as a real time game controller.

The neural network used is a standard multi-layer feedforward fully connected MLP with 10 inputs, a single hidden layer with 6 hidden nodes and 2 outputs. The inputs are the angle to the other car, the distance to the other car, the orientation of the other car, the angle to the current waypoint, the distance to the current waypoint, the angle to the next waypoint, the distance to the next waypoint, the direction of movement, the orientation of the car, and the speed of the car. Each neuron implements the hyperbolic activation function. At each time step, the observational inputs are fed from the sensor model to the neural network. The outputs are two real number values, one for steering and one for driving. Four additional threshold variables are defined to discretize the outputs into the on-off controllers of the simulator. The variables SteeringLimitLow and SteeringLimitHigh defines the lower and upper threshold of the steering output while the variable DrivingLimitLow and DrivingLimitHigh defines the lower and upper threshold for the driving output. The pseudo code for discretizing the output of the neural network can be summarized as follows.

```
if (output_steer < SteeringLimitLow)
    steering = 0;
elseif (steering > SteeringLimitHigh)
    steering = 2;
else
    steering = 1;
if (output_drive < DrivingLimitLow)
    driving = 0;
elseif (output_drive > DrivingLimitHigh)
    driving = 2;
else
    driving = 1;
```

Note that although SteeringLimitLow is logically supposed to be less than SteeringLimitHigh, it is not enforced as a constraint in the evolution so as to promote discovery of varied strategies. In a similar way, DrivingLimitLow and DrivingLimitHigh are not constrained in any way. For example, in the event that DrivingLimitLow is evolved to be a large positive number, the controller will likely drive the car in reverse most of the time. This representation of the controller is theoretically capable of driving either forwards or in reverse, and also to come to a complete stop. The neural network weights as well as its output threshold variables are trained using evolution strategies.

4.2.1.1 Experiments

A (100+100) evolution strategies (ES) [124], running for 200 generations was used as a training method for the neural network controller. The mutation operator was a Gaussian perturbation with the step size set to a fixed value of 0.1 for all variables. The evolution parameters are summarized in Table 4.1. Each individual is evaluated against the HSC for 5 rounds of competition, followed by 5 rounds of competitive co-evolution against another individual from the population. The competitive co-evolution was introduced to prevent over training, encourage better generalization and also to promote population diversity [53]. The fitness function was defined as the number of waypoints the individual collected averaged over the 10 rounds of game play. No solo game was used during the training. Elitism was implemented by retaining the best 4 individuals from each generation. Each chromosome for the neural network controller was encoded with a total of 84 real valued variables, 80 for the neural network weights and bias and 4 for output thresholds. In terms of computation time, each run of 200 generations, consisting of 200 × 100 × 10 = 200000 games took less than 10 minutes to complete.

The evolution of the neural network controller is plotted in Figure 4.1. It was observed that the neural network controller stagnated in terms of mean score at around 16 points after the first 30 generations. The comparative results, averaged over 500 games, for the neural network controller against the heuristic controllers are shown in Table 4.2. The mean results were given and the standard deviation quoted in parentheses.

In solo runs, it was observed that the neural network controller is able to outperform all 3 heuristic controllers with a higher mean score and yet smaller standard deviation. This implied that the evolved neural network controller is a well optimized and consistent driver. In competitive 2 player games against the heuristic controllers, the neural network controller was able to defeat all its heuristic controller opponents in mean score over 500 games. In particular, the game against the HCC yielded a very high combined end game score of 16.246 + 15.440 = 31.686. This was due to the waypoint predictive nature of the HCC. When the neural network controller was heading towards the current waypoint, there were instances where the HCC gave up the current waypoint and headed towards the next waypoint. So when the neural network controller passed the current waypoint, the HCC very quickly passed the newly activated waypoint. This made the game faster paced and led to high end game scores.

Since the neural network is a black box controller, its driving behaviour was analyzed visually. While the neural network was randomly initialized, it did evolve into a competent point to point driver that took advantage of the difference in acceleration between driving forward and in reverse. The neural network controller avoided the problem of orbiting a waypoint faced by the heuristic controllers by driving entirely in reverse. The lower acceleration gave it more control in steering and a smaller turning radius that improved its maneuverability and made navigating around waypoints an easier task. However, the novelty stopped at that.

Table 4.1 Evolution parameters for neural network controller

Parameter	Neural network
Method	Plus
Population size	100
Generations	200
Mutation type	Gaussian
Mutation probability	1
Mutation step size	0.1

Table 4.2 Results for neural network controller

Controller	Score
Greedy (GC)	12.774 (5.103)
HeuristicSensibleController (HSC)	10.578 (6.601)
HeuristicCombinedController (HCC)	9.284 (6.414)
Neural Network Controller (NN)	20.188 (3.090)
NN - GC	14.692 (2.092) - 11.020 (1.891)
NN - HSC	14.516 (2.240) - 11.996 (2.104)
NN - HCC	16.246 (3.393) - 15.440 (3.189)

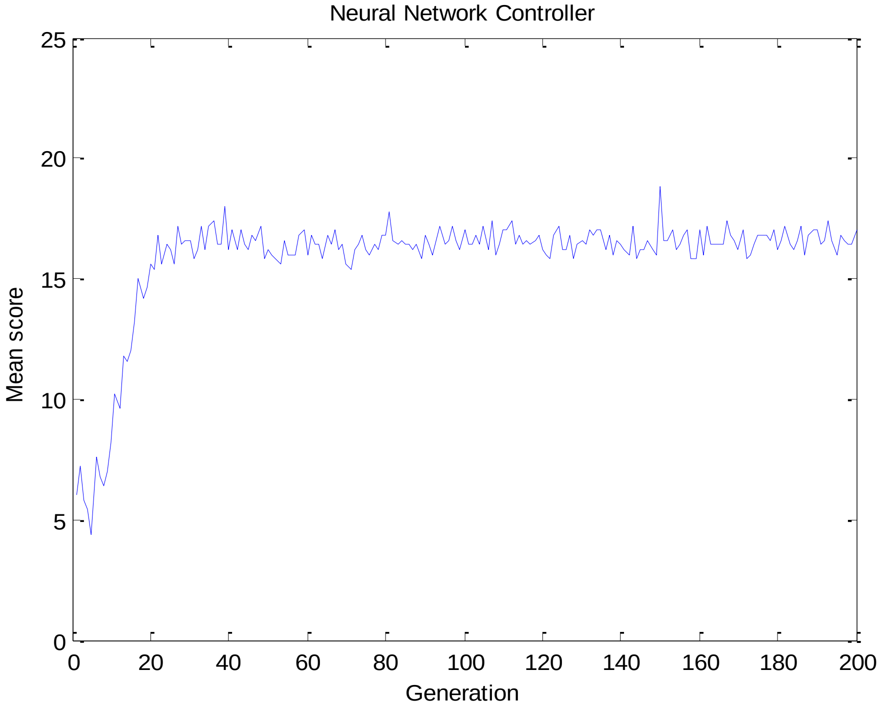


Figure 4.1 Training fitness of neural network controller

The neural network controller never learnt to decelerate or stop and hence always overshot the target point. The controller did not drive in the opposite direction (i.e. forwards) towards a waypoint. The neural network controller also did not evolve any form of prediction mechanism to decide to approach the next waypoint when the current waypoint was an obvious loss. Therefore, it was concluded that although the neural network representation used here does theoretically allow for the evolution of advanced driving behaviours and strategies, it only exploited the most basic of driving behaviours in the game and was trapped in a local minima. Perhaps the choice of using a single large neural network to approach this complex racing game was too ambitious. It may be possible to break down the individual aspects of driving and train separate neural networks to learn each part independently or in tandem. The lessons learnt from evolving the neural network controller was used to design the behaviour components of the behaviour-based controller in the next section.

4.2.2 Behaviour-based controller

The proposed controller design is inspired by the behaviour-based design methodology to take advantage of its computation efficiency. In this section, the behaviour-based AI (BBAI) methodology will be briefly described followed by a detailed discussion of the various components and performance of the behaviour-based controller.

In BBAI, intelligence is made up of a large number of modular components which are relative simple and robust. Each of these components work within a specific set of conditions which it is able to observe from the environment. These components are organized into layers in a hierarchy which are able to interact with one another. The constraints here are that no components will have access to another’s internal states but it is possible to observe their inputs and outputs. Additionally, a higher layer may subsume a lower layer by affecting its inputs and outputs. This is known as the subsumption architecture [22].

This methodology is powerful because of its simplicity and robustness. Each individual component encodes only a simple behaviour such as moving forward, turning or avoiding objects, and thus can be programmed reliably. In the original implementation, there was no memory or learning in its architecture, hence the resultant was that of a reactive behaviour. This turns out to be an advantage for BBAI because it is computationally efficient and hence suitable for systems that require good response time such as in the design of smart robots [42] [122] and also in the real time racing game in this chapter.

In the behaviour-based controller, the basic driving behaviours of the car racing controller such as accelerating, braking and steering are organized at the lowest level. A separate component for the prediction of waypoints is placed at a higher level so that it is able to augment the input of the driving layer in order to dictate which waypoint is more advantageous for the car to drive towards.

The disadvantage of reactive intelligence is its design process because it performs no search or learning by itself [26] and all behaviours must be designed by hand. However, this difficulty can be adverted with the inclusion of computational intelligence as a design companion. In the behaviour-based controller, only generic representations are specified for each behaviour component. Each component is subsequently trained using genetic algorithm. For example, a potential field representation is used for the steering control but it is not specified beforehand whether the interactions are attractive or repulsive in nature. The evolved controller exhibits attractive forces which is necessary for good performance in the game.

The behaviour-based controller consists of 5 main components. The interactions between components are illustrated in Figure 4.2. Components that form the basic driving behaviour are organized at the lower base level while tactical behaviours are organized at the higher first level. The first level can be said to subsume the base driving level. The individual component will now be discussed in further detail.

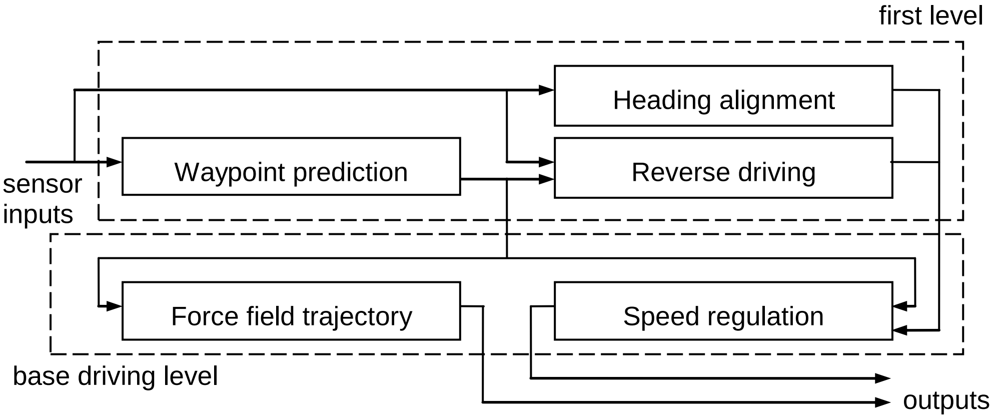


Figure 4.2 Overview of behaviour-based controller

The advantage of the behaviour-based methodology over the neural network one is that the former is a white box design while the latter is a black box design. Being a white box allows the designer to gather insights to how individual components complement one and another, and how parameters are evolved to exhibit the winning behaviour. Furthermore, the behaviour-based methodology allows the designer to input domain knowledge which can guide the evolution towards better solutions with faster convergence.

4.2.2.1 Force field trajectory

The first is a trajectory planning mechanism inspired by the interaction between charged particles in space. Potential field methods are widely used in the field of robotics due to its simplicity [125]. Every foreign particle in the playing area, namely the car belonging to the opponent, the current waypoint and the next waypoint, induces either an attractive or repulsive field on the game area. At any point in the game area, the controller tries to align its car in the direction of the local induced vector field. As such, the controller car will move along the resultant field lines induced by the interaction of these charged particles. However, these field lines only indicate the steering path and not the driving speed. The field equation for the particles in the game arena is defined in (4.1).

$$\vec{E}_i = q_i r^{p_i} \hat{r}\,,\ i = \{other, wp1, wp2\} \tag{4.1}$$

where other is the opponent vehicle, wp1 is the current waypoint, wp2 is the next waypoint, $\vec{E}_i$ is the field vector induced by the point particle i, q_i is the charge of particle i, r is the distance from the particle with charge q_i to the evaluation point, p_i is the power factor of the distance r and $\hat{r}$ is the unit vector pointing from the particle with charge q_i to the evaluation point. The variables q_i and p_i for the opponent car, the current waypoint and the next waypoint are optimized using genetic algorithm. The controller car is considered a positive point charge in calculations in order to evaluate the resultant force exerted on the controller car. There are no constraints on the evolved variables so it is entirely possible that the results may turn out to be other than expected.

4.2.2.2 Speed regulation

The force field trajectory component determines only the driving path of the car and not the driving speed. Hence, a speed regulating function, which constitutes the second base component of this controller, is introduced to specify the driving speeds along the steering trajectory. An important driving feature which is crucial for the performance of the controller is the ability to stop at a specific position in the playing area. Although it may seem counterintuitive to stop in a racing game, this action becomes necessary when one considers going for the next waypoint instead of the current waypoint. Suppose the opponent is going to reach the current waypoint first, it makes sense to head towards the next waypoint directly. But in the situation that the controller car arrive at the next waypoint before the opponent can reach the current waypoint, the controller will then need to stop the car at the next waypoint and wait until it becomes activated. The equation for the speed regulating function is defined in (4.2).

$$Speed = a \times \tanh\left(b \times r + c\right) + d \tag{4.2}$$

where r is the distance to the destination and a, b, c and d are parameters characterizing the speed regulation function respectively. The 4 parameters are optimized using genetic algorithm. The hyperbolic tangent function is chosen because of its general shape. The tapering of its outputs at high values of r is analogous to the notion that the car should cruise at a constant speed at far distances from its destination (i.e. the cruising speed should not increase indefinitely with distance). Additionally, the steep gradient around the origin is analogous to deceleration when it is near the destination. The values a, b, c and d serves to shape the hyperbolic tangent function to one most desirable for this car racing simulation. There are no constraints that the function needs to pass through the origin or that it should be positive or negative.

4.2.2.3 Reverse driving

A desirable driving feature for this type of point to point race is the ability to drive in reverse. A human player who just started playing the game will very soon realize that if a waypoint is situated at close proximity but directly behind the car, it is faster to simply reverse the car towards it. This type of behaviour was not present in the HeuristicSensibleController (HSC). As a result, the HSC was often seen, much to the frustration of the human observer, to take a non-optimal U-turn to approach a waypoint behind it. Moreover, in the process of performing the U-turn, the controller often underestimated the turning radius and became trapped in an endless orbit about the waypoint. To rectify such unrealistic behaviours, a reverse driving threshold variable is introduced to the behaviour-based controller. The angle towards the destination is included to determine whether to drive forwards or in reverse for a given situation. If the angle is within a given threshold, the speed function will be negated and the controller will reverse the car towards the destination instead. The threshold parameters are also evolved using genetic algorithm.

4.2.2.4 Waypoint prediction

The fourth component is a predictive module that chooses which waypoint to compete for. By observing the state of the game area, the controller 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 towards the next waypoint instead and vise versa. This predictive module sits on top of the base driving layer and is capable of augmenting the inputs to the base driving layer. The pseudo code for the waypoint prediction component is as follows.

```
if $\left(\frac{distance\,(c,wp1)}{distance\,(o,wp1)} < 1\right)$
        return c;
elseif $\left(\frac{distance\,(c,wp1)}{speed\,(c,wp1)} \times \frac{speed\,(o,wp1)}{distance\,(o,wp1)} < 1\right)$
        return c;
else
        return o;
end
```

where distance (i, j) refers to the Euclidean distance measured between point i and point j, speed (i, j) refers to the magnitude of the vector component of the speed of vehicle i along the direction from point i towards point j, c is the car controlled by the controller calling this function, o is the opponent vehicle and wp1 is the current waypoint.

During each time step, the waypoint prediction system determines which vehicle will reach the current waypoint first. If the opponent vehicle will reach first, then the controller will direct both the force field trajectory and the speed regulator towards the next waypoint instead. The waypoint prediction system is designed using simple domain knowledge and reasoning. First, the component speed of each vehicle in the direction of the current waypoint is calculated using vector scalar product. Next, the pseudo code is used to determine which vehicle will reach the current waypoint earlier. In essence, the controller will drive the car towards the current waypoint if it is nearer to the current waypoint than the opponent vehicle is. Even if it is further away compared to the opponent, it will still drive towards the current waypoint if it takes a shorter time to reach there based on the instantaneous component speed of each vehicle calculated in the previous step.

4.2.2.5 Heading alignment

When the controller is driving the car towards the current waypoint, it may be advantageous to align its heading to that of the next waypoint just as it passes through the current waypoint. This will allow for a smoother driving line from the current waypoint to the next and increase the likelihood of reaching the next waypoint before its opponent. Currently, this behaviour is only implemented for the forward driving direction. The pseudo code for the heading alignment behaviour is as follows.

```
if $\left(\begin{array}{c}\left(distance\left(c,wp1\right)<k_1\right)AND\left(-k_2<angle\left(c,wp2\right)<k_2\right)\\ AND\left(speed\left(c\right)>k_3\right)\end{array}\right)$
        return true;
else
        return false;
end
```

where distance (i, j) refers to the Euclidean distance measured between point i and point j, angle (i, j) refers to the angle in radians of point j from point i, speed (i) refers to the current instantaneous speed of object i, c is the car controlled by the controller calling this function, wp1 is the current waypoint and wp2 is the next waypoint. Three variables k_1, k_2 and k_3 defines the thresholds of the activation of this function and are evolved using genetic algorithm. If the function returns true, the car will be steered to face the next waypoint instead of the current waypoint. Otherwise, the car will continue on its current path.

4.2.2.6 Experiments

A (50+50) ES, running for 200 generations was used as a training method for the behaviour-based controller. As the behaviour-based controller had lesser variables in its chromosome, it was trained with a reduced population size of 50, all other conditions remained constant as with the neural network controller. The mutation operator was a Gaussian perturbation with the step size set to a fixed value of 0.1 for all variables. Each individual was evaluated against the HSC for 5 rounds of competition, followed by 5 rounds of competitive co-evolution against another individual from the population. The fitness function was defined as the number of waypoints the individual collected averaged over the 10 rounds of game play. Elitism was implemented by retaining the best 4 individuals of each generation. Each chromosome for the behaviour-based controller was encoded with a total of 14 real valued variables, 6 from force field component, 4 from speed regulation component, 1 additional variable which encoded the threshold for reversing driving, and 3 variables for the heading alignment component. All variables were initialized by a random Gaussian distribution with a mean of 0 and a variance of 1. The parameters are summarized in Table 4.3.

The force field trajectory component provided some guidelines for the controller to plan its path from the current waypoint to the next waypoint. The speed regulating component defined the acceleration, deceleration and stopping behaviour. The reverse threshold decided when the best time to reverse towards a target was. The waypoint prediction directed the car to the next waypoint if the current waypoint cannot be reached before the opponent. Finally, the heading alignment component made sure the car will be in a good position for the next waypoint. The evolution of the behaviour-based controller is plotted in Figure 4.3. It was noted that potential field methods such as the force field trajectory component used here were prone to the problem of local minima. However, this problem was not noticeable in the experiments as little variations were observed when parameters were tuned.

From the simulation results, it was observed that a driving speed limit of approximated 7 units per time step was imposed by the speed regulating function on the controller vehicle. For distances less than 0.2 to the destination, the car switched to rapid deceleration before coming to a halt at the destination point. The negative values of distance r were not used in the actual game as distances were strictly positive. Traveling within these speed limits, the car could not skid and hence did not exhibit any advanced driving techniques that required skidding.

Table 4.3 Evolution parameters for behaviour-based controller

Parameter	Behaviour-based
Method	(50+50)
Population size	50
Generations	200
Mutation type	Gaussian
Mutation probability	1
Mutation step size	0.1

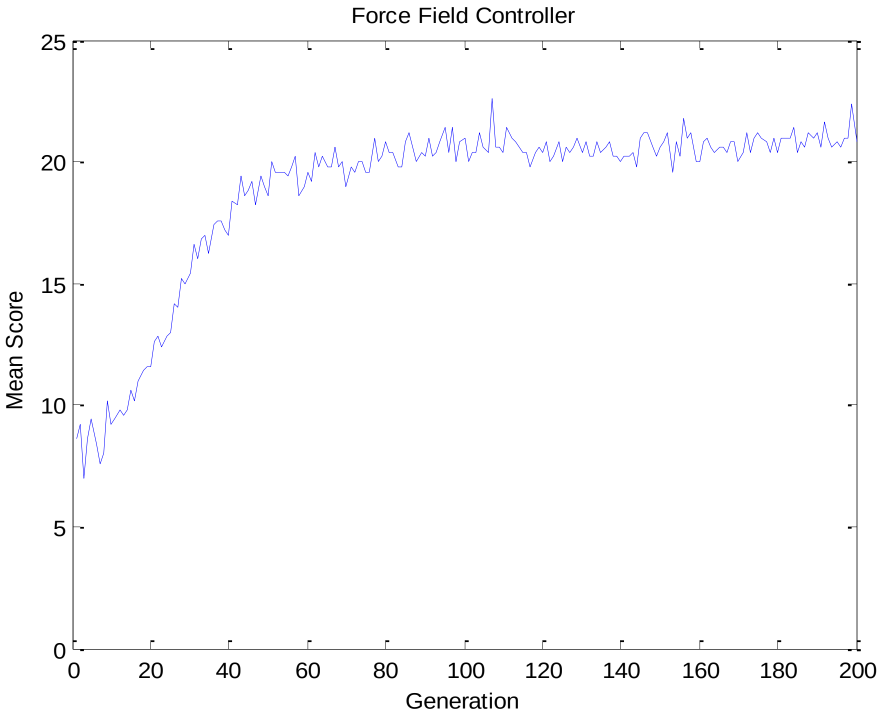


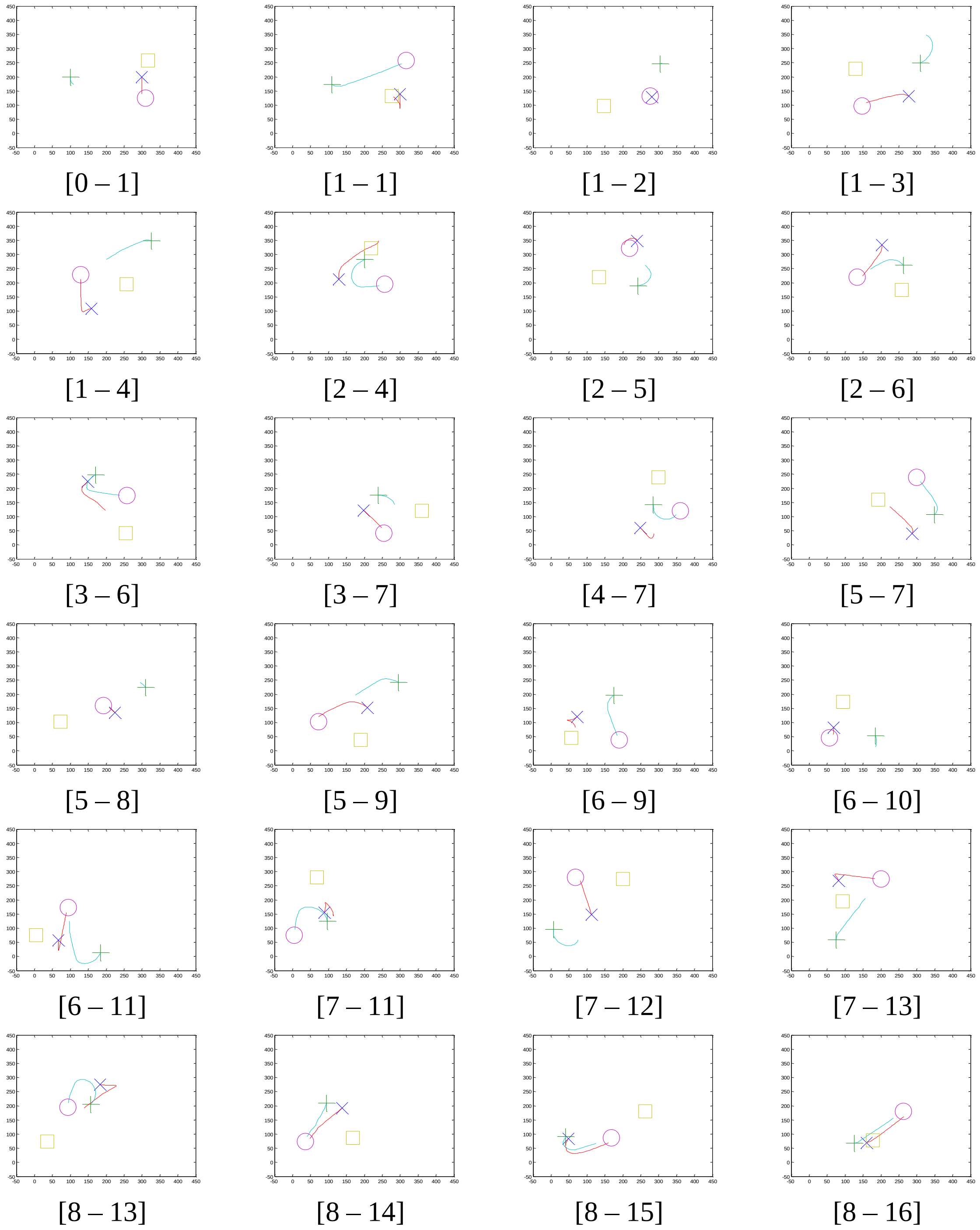
Figure 4.3 Training fitness of behaviour-based controller

An additional parameter was included to determine whether to drive forwards or in reverse for a given situation. If the angle to the destination was within the threshold stated by the parameter, the speed regulating function would be negated and the controller would reverse the car towards the destination instead. The final evolved value of this parameter was 1.897 radians. This implied that if the waypoint was located within a span of 142.5° centred directly behind the car, the controller would drive in reverse towards the destination instead.

4.2.3 Comparative discussion

The overall performance of the behaviour-based controller and its comparison against the neural network controller will be presented in this section. The results of the comparative studies from both controllers are summarized in Table 4.4. Parts of the results were retrieved from Table 4.2

and presented here for better readability. The mean results were given and the standard deviation presented in parentheses.



[0 – 1] [1 – 1] [1 – 2] [1 – 3]

[1 – 4] [2 – 4] [2 – 5] [2 – 6]

[3 – 6] [3 – 7] [4 – 7] [5 – 7]

[5 – 8] [5 – 9] [6 – 9] [6 – 10]

[6 – 11] [7 – 11] [7 – 12] [7 – 13]

[8 – 13] [8 – 14] [8 – 15] [8 – 16]

A circle symbol marks the current waypoint; a square symbol marks the next waypoint; a plus symbol and a cross symbol respectively marks the starting position of the neural network and the behaviour-based controller when a new waypoint is activated; solid lines marks the paths traced by the respective controllers. Each sub-diagram ends when one of the controllers passes the current waypoint, and is annotated by the score of the game just after the current waypoint is passed. The score is read as [neural network – behaviour-based].

Figure 4.4 Point by point diagram of a partial game between neural network controller and behaviour-based controller

Table 4.4 Comparative results between neural network controller and behaviour-based controller

Category	Neural network	Behaviour-based
Training best	18.0	22.6
Solo	20.188 (3.090)	22.626 (2.429)
Time taken	76.1 (3.035) seconds	36.1 (0.316) seconds
vs GC	14.692 (2.092) - 11.020 (1.891)	18.630 (2.047) - 10.266 (2.034)
vs HSC	14.516 (2.240) - 11.996 (2.104)	18.726 (2.023) - 11.194 (2.283)
vs HCC	16.246 (3.393) - 15.440 (3.189)	19.388 (4.379) - 15.108 (3.663)
vs each other	13.826 (2.391)	20.324 (1.895)

In the solo game, the behaviour-based controller obtained a mean score of 22.626 which outperformed the neural network controller's score of 20.188, averaged over 500 trials. At the same time, the behaviour-based controller also had a smaller standard deviation implying that it is more competent and also more consistent compared to the neural network controller. Similarly, in 2 player competitions against the 3 heuristic controllers, the behaviour-based controller was able to achieve larger winning margins as well as higher nominal mean scores. The two controllers were also placed in a direct competition with each other to validate their relative performance with respect to each other. In direct competition, the behaviour-based controller scored 20.324 points against the 13.826 points of the neural network controller. A visual inspection of the match up was conducted to further ascertain the reasons for the behaviour-based controller's better performance. It was observed that the reverse driving and waypoint prediction components were the main contributors to the success of the behaviour-based controller. The reverse driving decision component was able to choose the more time efficient route for the controller to approach its target, as observed in Figure 4.4 [6-11] and [7-13]. This could be seen from the sharp angles in the paths traced by the cross symbol (behaviour-based controller), while the plus symbol (neural network controller) was seen making large U-turns in Figure 4.4 [2-4], [6-11] and [7-12]. The waypoint prediction component enabled the controller to quickly reclaim a point by driving to the next waypoint when the current one cannot be won, as observed in Figure 4.4 [2-4], [5-7] and [6-9]. In particular, the cross symbol (behaviour-based controller) traced a path towards the square symbol (next waypoint) when it predicted a loss for the current waypoint. These features made the driving path traced by the behaviour-based controller during the game very fluid and efficient.

A performance indicator raised earlier in this chapter was the computational efficiency of the controller. A game AI in a real time driving game such as this would not have the computation resource to evaluate all possible moves at a given game state. Both the neural network and the behaviour-based design methodology were considered because of their computationally efficient characteristics. To investigate their comparative time efficiency, both controllers were timed for 10 sets of 5000 solo run trials and the results are also shown in Table 4.4. To establish a common benchmark for comparison, all simulations were conducted on the same computer terminal under the same boot conditions. The neural network controller took 76.1 seconds to complete 5000 solo run trials or 76.1 / 5000 = 0.0152 seconds per trial or 0.0152 / 1000 = 15.2 microseconds per time step while the behaviour-based controller took 36.1 seconds for 5000 trials or 0.00722 seconds per trials or 7.22 microseconds per time step. To put the comparison into perspective, in a visual game where the car racing game is graphically simulated on screen, each game typically lasts 60 seconds. Both controllers were computationally efficient but the behaviour-based controller was able to outperform the neural network controller while being computationally 2.11 times faster. The computational gain may not be significant in this simplified simulation as there was no emphasis on graphics and sounds. However, in the context of a commercial game where a large percentage of the CPU cycle is dedicated to rendering graphics and sounds, a computationally efficient game controller becomes desirable. Also, in this simulation, there are only 2 game agents present. In games where there are hundreds of interacting game agents, the savings in computational time becomes significant in ensuring an uninterrupted game presentation.

4.3 Results and analysis

In this section, the behaviour-based controller will be analyzed with respect to the effects of crossover and mutation, the parameters evolved, the performance of individual behaviour components, and the generalization performance against opponents unseen during training.

A genetic algorithm (GA) of population size of 30, running for 100 generations was used as a training method for the behaviour-based controller. Each individual was evaluated against the HeuristicSensibleController (HSC) for 5 rounds of competition, followed by 5 rounds of competitive co-evolution against a random elite individual from the population. The fitness function was defined as the number of waypoints the individual collected averaged over the 10 rounds of game play. Each game was played for 1000 time steps. Elitism was implemented by retaining the best 4 individuals of each generation. The same 4 elite individuals also participate as co-evolution opponents during the fitness evaluation of other individuals in the population. Each chromosome for the behaviour-based controller was encoded with a total of 14 real valued variables, 6 from force field component, 4 from speed regulation component, 1 variable which encoded the threshold for reversing driving, and 3 variables for the heading alignment component. All variables were initialized by a random Gaussian distribution with a mean of 0 and a variance of 1. The crossover and mutation operate will be discussed in further details next.

4.3.1 Effects of crossover operator

Each chromosome was encoded using real values so the crossover operator must be designed to work with real numbers. The pseudo code for the crossover operator is as follows.

```
for each pair of genes
    if (random [0,1] < crossover rate)
        weight = random [0,1];
        offspring = weight × parent1 + (1 – weight) × parent2;
    end
end
```

The variable weight placed a random emphasis on the gene from one parent. For example, if the weight was 0.5, the result would be the average value of the genes from both parents. However, if the weight was 0.8, then the offspring would inherit 80% of the gene from parent1 and the remaining 20% of the gene from parent2.

The effect of the crossover rate was investigated by varying its value from 0.0 to 1.0 in steps of 0.2 while the value of the mutation rate was arbitrarily set to 0.2. The results of varying the crossover rate are plotted in Figure 4.5. The inclusion of the crossover operator generally produced better results compared to when the crossover rate was set to 0. In all cases, the results converge after about 30 generations.

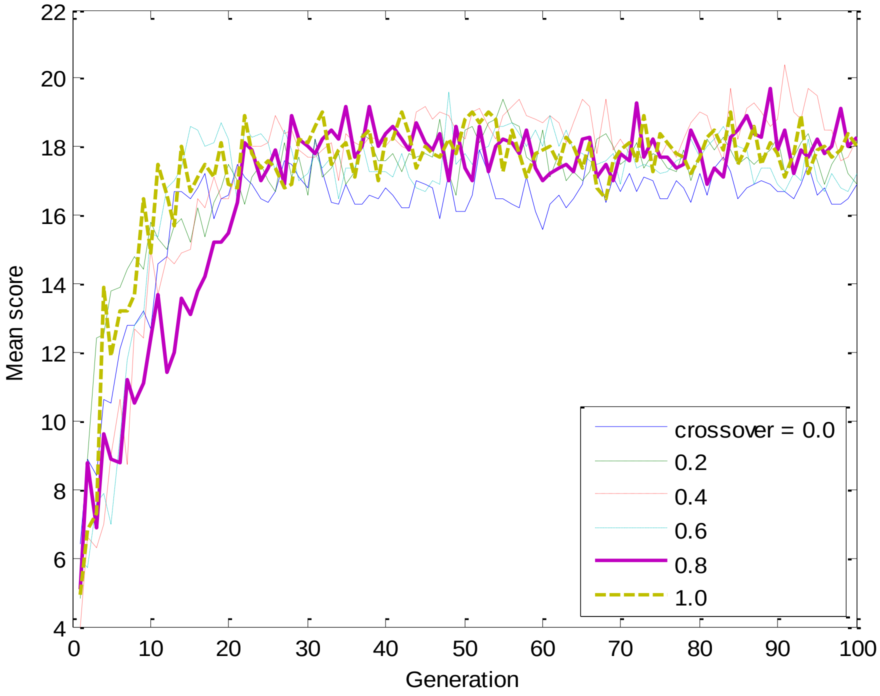


Figure 4.5 Effects of varying crossover rate; mutation rate fixed at 0.2

4.3.2 Effects of mutation operator

A Gaussian perturbation with a mean value of 0 and variance of 1 was used as the mutation operator. For each gene, the Gaussian perturbation was applied with a probability given by the mutation rate.

The effect of the mutation rate was investigated by varying its value from 0.0 to 1.0 in steps of 0.2 while the value of the crossover rate was arbitrarily set to 0.8. The results of varying the mutation rate are plotted in Figure 4.6. In particular, the case of mutation rate = 0.0 converged to a local minimum, likely due to the lack of diversity. The value of mutation rate = 0.2 was observed to be most optimal amongst the different choices of mutation rate. The rate of convergence was also around 30 generations.

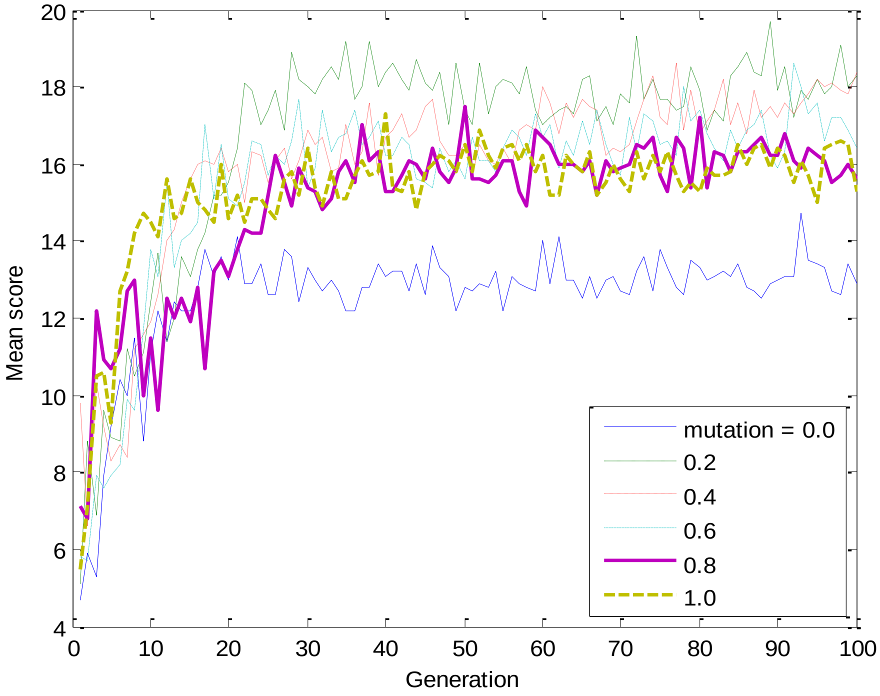


Figure 4.6 Effects of varying mutation rate; crossover rate fixed at 0.8

4.3.3 Analysis of evolved parameters

Based on the prior investigation, a crossover rate of 0.8 and a mutation rate of 0.2 were chosen to evolve the behaviour-based controller. The best individual from the last generation was examined to investigate the characteristics of the evolved behaviour-based controller. The 5 components are described as follows. The force field trajectory component provided some guidelines for the controller to plans its path from the current waypoint to the next. The speed regulating component defined the acceleration, deceleration and stopping behaviour. The reverse threshold decided when the best time to reverse towards a target is. The waypoint prediction directed the car to the next waypoint if the current waypoint cannot be reached before the opponent. Finally, the heading alignment decided when it is best to turn towards the next waypoint. The results and performance of the behaviour-based controller will be discussed from 2 perspectives in this section. Firstly, the white box nature of the behaviour-based controller allowed for the analysis of the evolved values. These evolved parameters will be examined to gain a better appreciation of the behaviour of the behaviour-based controller. Secondly, the functionality of the individual components will be examined for their impact on the overall performance of the controller.

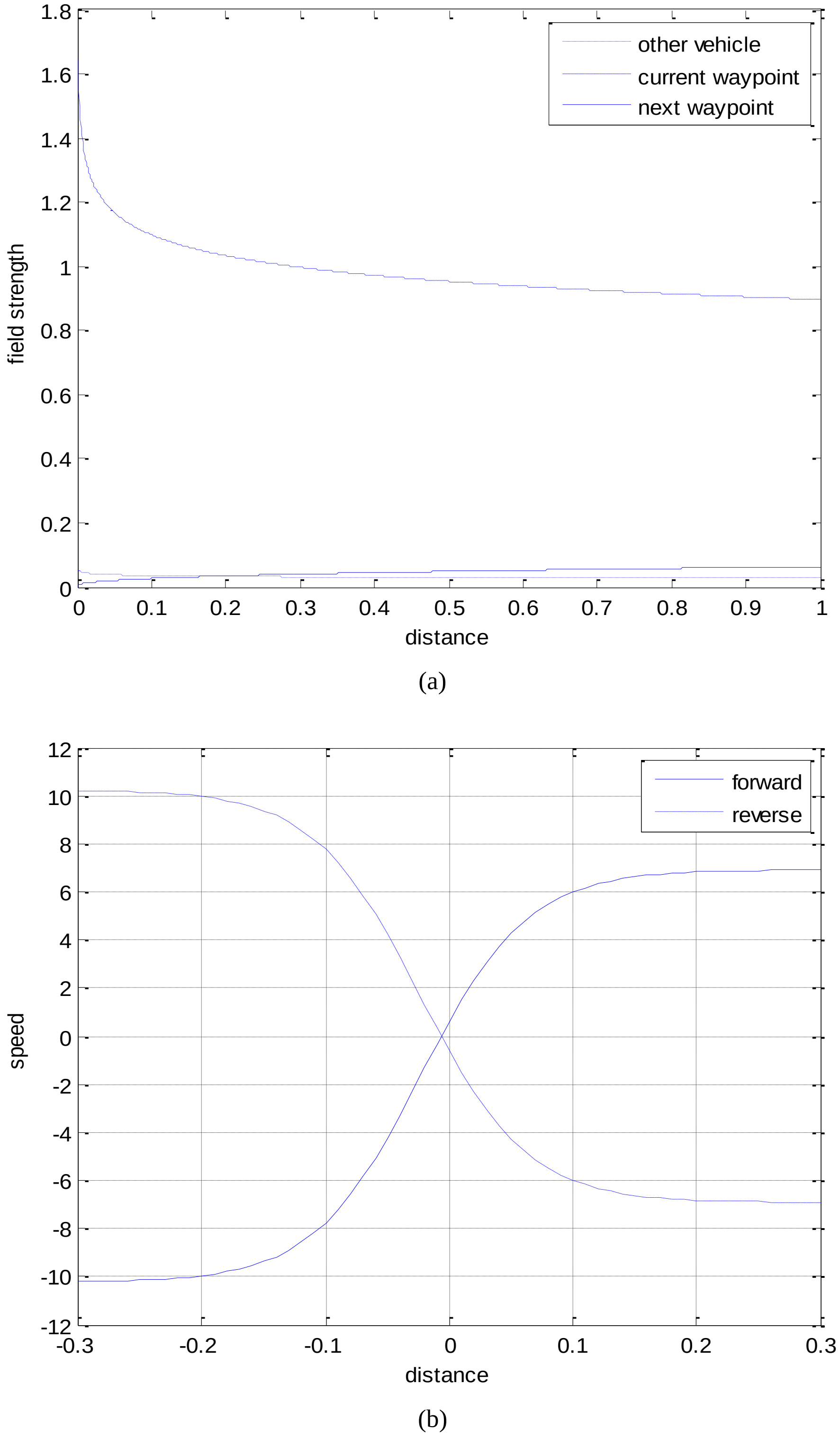
The evolved values for the force field trajectory component are presented in Table 4.5 and the field strength is plotted against distance r in Figure 4.7. All forces acting on the car were attractive in nature since all the controlled cars were assumed to be a positive point charge and the evolved q_i values turned out to be negative. The field strength of the current waypoint was at least 10 times larger than that of the opponent car and the next waypoint within the range of the game area. This implied that the controller car was strongly attracted to the current waypoint while the effects from the opponent car and the next waypoint were minimal. Therefore, the controller would direct the car towards the current waypoint regardless of its distance. This result was similar to the common intuition that is to steer in the direction of the destination. Additionally, this way of steering was applicable both when driving forward and in reverse. The value of q_{other} was initially expected to be repulsive in nature as it seemed sensible to avoid collisions with the opponent, but this controller evolved a new strategy that was to intentionally collide with the opponent when sufficiently near. This was because the reverse driving component allowed the controller to recover quickly after a collision by simply driving in the direction facing the current waypoint. This became an advantage if the opponent only drove in one direction like the HSC. In general, the output trajectory of the force field component was an approximate straight line towards the current waypoint with minor disturbances coming from the other vehicle and next waypoint.

Table 4.5 Evolved force field trajectory parameters of best individual

i	other	wp1	wp2
q_i	-0.02803	-0.896679	-0.063289
p_i	-0.10153	-0.08817	0.377045

From Figure 4.7 (b), it was observed that a driving speed limit of approximately 7 units per time step was imposed by the speed regulating function on the controller vehicle. For distances less than 0.2 to the destination, the car switched to rapid deceleration before coming to a halt at the destination point. This could be seen by the speed regulation function passing through the origin. The negative values of distance r were not used in the actual game as distances were strictly positive in the game. Traveling within these speed limits, the car could not skid and hence did not exhibit any advanced driving techniques that required skidding.

An additional parameter was included to determine whether to drive forwards or in reverse in a given situation. If the angle to the destination was within the threshold stated by the parameter, the speed regulating function would be negated and the controller would reverse the car towards the destination instead. The final evolved value of this parameter was 1.897 radians. This implied that if the waypoint was located within a span of 142.5° centred directly behind the car, the controller would drive in reverse towards the destination. Additionally, the speed regulating function plots of both forward and reverse in Figure 4.7 (b) passes very close to the origin. In the actual game, this was sufficient to stop the vehicle exactly at its desired destination. This was observed in two player competition where the behaviour-based controller often stopped at the next waypoint while waiting for the current waypoint to be passed by its opponent.



(a)

(b)

Figure 4.7 Graph of evolved parameters for behaviour-based controller for (a) field strength against distance from particle and (b) desired driving speed against distance from destination

Table 4.6 Evolved heading alignment parameters of best individual

Parameter	Value
k_1	0.0551
k_2	2.5513
k_3	4.3076

The evolved parameters for the heading alignment component are summarized in Table 4.6. At a distance of 0.0551, the behaviour-based controller was likely in a state of deceleration according to Figure 4.7 (b). At this distance, if the next waypoint was within -2.5513 to 2.5513 radians or within -146.2° to 146.2° centred in front of the car and the speed of the car was more than 4.3076 units per time step, the heading alignment component would steer the car to face the next waypoint while continuing its approach toward the current waypoint. The speed threshold ensured that the car did not miss its current waypoint while trying to steer towards the next. This enabled the behaviour-based controller to put itself in a better position to accelerate towards the next waypoint once it passed the current one.

4.3.4 Analysis of behaviour components

Five behavioural components were implemented in the behaviour-based controller. Each of these components could be optionally activated or deactivated, giving a total of 32 combinations. In order to appreciate the impact of each component on the overall performance of the behaviour-based controller, all combinations of the controller were benchmarked against the case of solo run, competition against the HeuristicSensibleController (HSC) and against the HeuristicCombinedController (HCC) in Table 4.7. The combination of components activated is abbreviated under the column Behaviour in the format $X_1X_2X_3X_4X_5$ where X_1 represents waypoint prediction, X_2 represents force field trajectory, X_3 represents speed regulation,

X_4 represents reversing and X_5 represents heading alignment. In the deactivated state, the force field trajectory was replaced with intuitive steering that steered in the direction of the destination and the speed regulation threshold was set to 7 independent of distance. As a reference, the results for HSC running the same benchmark are also listed on row 0.

It was observed that component X_1 (waypoint prediction) had insignificant impact during the solo run. This was to be expected as there was no opponent in the solo case and performance was entirely dependent on driving behaviours. In two player situations, waypoint prediction generally improved the controller performance as evident when comparing pair wise between rows 1 to 16 with their counterpart from rows 17 to 32. This implied that waypoint prediction mainly contributed to improvements in competitive games.

Although pair wise comparisons for X_2 (force field trajectory) from Table 4.7 did not give a clear indication of its advantage, its value could be better appreciated visually. In general, the driving line traced by the force field trajectory was smoother than that of intuitive steering, resulting in a more realistic driving style rather than a mechanic one that constantly jerked left and right in order to keep on a straight path.

By comparing pair wise of rows 1 & 5 and other corresponding pairs that compare X_3 (speed regulator), it was observed that the speed regulator improved performance both in the solo run and against HCC. This was mainly due the speed regulator slowing the car down near its destination, hence greatly reducing the occurrence of orbiting, and this translated into higher scored points for the controller. The difference was even greater when the speed regulator worked in combination with the waypoint predictor as it enabled the controller to stop the car at the next waypoint when waiting for its opponent to pass the current waypoint.

Table 4.7 Comparative studies of behaviour set

Row	Behaviour	Solo	vs HSC	vs HCC
0	HSC	13.730	10.528	10.726
1	00000	11.424	11.674	9.678
2	00001	10.802	11.668	9.660
3	00010	14.350	12.880	12.654
4	00011	14.016	12.766	12.580
5	00100	16.422	11.894	14.700
6	00101	17.178	12.236	15.256
7	00110	20.092	13.154	16.746
8	00111	20.974	13.272	16.858
9	01000	10.988	11.572	9.292
10	01001	11.308	11.684	9.774
11	01010	14.354	13.168	12.154
12	01011	13.854	13.044	12.324
13	01100	16.262	11.810	15.176
14	01101	17.532	11.922	15.204
15	01110	20.168	13.478	16.920
16	01111	21.006	13.388	16.938
17	10000	11.396	14.728	10.524
18	10001	10.926	14.828	10.574
19	10010	14.288	16.144	13.692
20	10011	13.904	16.216	13.442
21	10100	16.598	15.966	18.292
22	10101	17.472	16.184	18.970
23	10110	20.084	17.252	19.740
24	10111	20.776	17.646	20.116
25	11000	11.142	14.600	10.282
26	11001	11.248	14.672	10.658
27	11010	14.640	15.942	13.302
28	11011	13.646	16.116	13.374
29	11100	16.596	16.006	18.384
30	11101	17.182	16.046	18.316
31	11110	20.224	17.148	19.742
32	11111	21.074	17.758	20.026

In the comparison for component X_4 (reversing) the general trend observed was that activating the reverse driving feature improved performance in all three cases of solo run and competitive games (i.e. rows 1 & 3). The difference was more significant when it was used in combination with the speed regulator (i.e. rows 1 & 7). This was because the speed regulator slowed down the car at its destination, making the change of direction smoother and less time consuming. In visually observed games, the reversing behaviour was also seen as the main contributor to collision recovery as the controller was able to drive in whichever direction that was facing its destination after a collision.

The independent effects of component X_5 (heading alignment) could be observed by comparing the columns of solo play on rows 5 & 6 and rows 7 & 8. The heading alignment mainly improved the performance of solo games when used in conjunction with the speed regulation component. This was likely due to the fact that all parameters were evolved simultaneously. As a result, the genetic algorithm successfully exploited this collaboration between the two components. The heading alignment behaviour also improved results in two player games but the improvement was of a smaller margin.

The analysis of the individual components of the behaviour-based controller had also provided some insights on how the controller can be improved in the future. These suggestions will be discussed here for possible implementation in the future. An inefficiency of the speed regulator was that it treated the current waypoint and next waypoint indifferently, which turned out to be sub-optimal. Although the controller needed to stop the car at the next waypoint, the same cannot be said about the current waypoint. In a race to the current waypoint, there was no need to slow to a complete halt at the waypoint. Instead, crossing the current waypoint with moderate and controllable speed could be considered a better choice. Hence, separate speed regulation models for the current and the next waypoint could improve the performance of the controller. To further complicate matters, the forward acceleration was twice that of the reverse acceleration in the game. This meant the speed regulation could be further broken down into forward and reverse components rather than simply negating one component to get the other. While these incremental improvements will likely be beneficial, including these features will require detailed analysis of the game dynamics. Another method of implementation may be to divide the speed regulation component into 4 separate functions and employ GA to optimize the function parameters without the need to analyze the game dynamics.

4.3.5 Generalization performance

The comparative studies so far were conducted under a controlled environment where simple heuristic controllers were used. To further substantiate its performance, the behaviour-based controller was tested against the top 5 entries of the Simulated Car Racing Competition held during the 2007 IEEE Congress of Evolutionary Computation (CEC) [180] to test its generalization performance against previously unseen opponents. In the competition, each entry is ranked using a competition benchmark known as CompetitionScore. However, the controller with the highest benchmark score at this point is not necessarily the winner of the competition. The winner of the competition is the winner of a final round robin tournament. The tests in this section will be conducted in a similar manner. The top 5 controllers and the behaviour-based controller will run the benchmark CompetitionScore and their scores and the time taken for simulation will be recorded. Thereafter, all 6 controllers will take part in a round robin tournament and the scores of each pairings will be recorded.

4.3.5.1 CompetitionScore benchmark

The CompetitionScore metric is the benchmark metric used to rank the submitted controllers before the final tournament. This metric is defined as the mean fitness of 500 trials in each of these three scenarios: solo trial, versus HeuristicSensibleController (HSC) and versus HeuristicCombinedController (HCC). In order to achieve a high CompetitionScore, a controller needs to perform well on its own (i.e. solo run), as well as against a weak (i.e. HSC) and an intermediate (i.e. HCC) controller.

The mean scores and times of the CompetitionScore benchmark, and their standard deviations in parentheses, are presented in Table 4.8. Controllers A, B, C, D & E are the controllers ranked 1, 2, 3, 4 & 5 respectively on the competition website [177]. Since there are two metrics of comparison, the results are plotted in the form of a Pareto dominance diagram in Figure 4.8 for better visualization. The axes are shown in logarithmic form due to the presence of very large and very small differences in simulation times. The behaviour-based controller is highlighted in bold. The performance of the behaviour-based controller will be discussed in this section in terms of its score as well as its computation efficiency. The results are averaged over 10 runs.

The behaviour-based controller scored 19.558 in the benchmark and was ranked second amongst the 6 controllers. In terms of computation efficiency, the behaviour-based controller was the most efficient controller, completing the benchmark in 17.5 seconds. Comparatively, the top scoring controller (Controller A) took an average of 8450.1 seconds to complete the benchmark, or (8450.1 / 17.5 $\approx$) 482 times slower than the behaviour-based controller. In real time games, game AI is usually allocated a very small CPU cycle budget (a large portion goes to rendering graphics), making computational intensive algorithms less attractive and hence the need for an efficient game controller. This makes the behaviour-based controller a more suitable candidate for implementation in real time games.

Table 4.8 Comparative results of CompetitionScore of behaviour-based controller against top 5 controllers

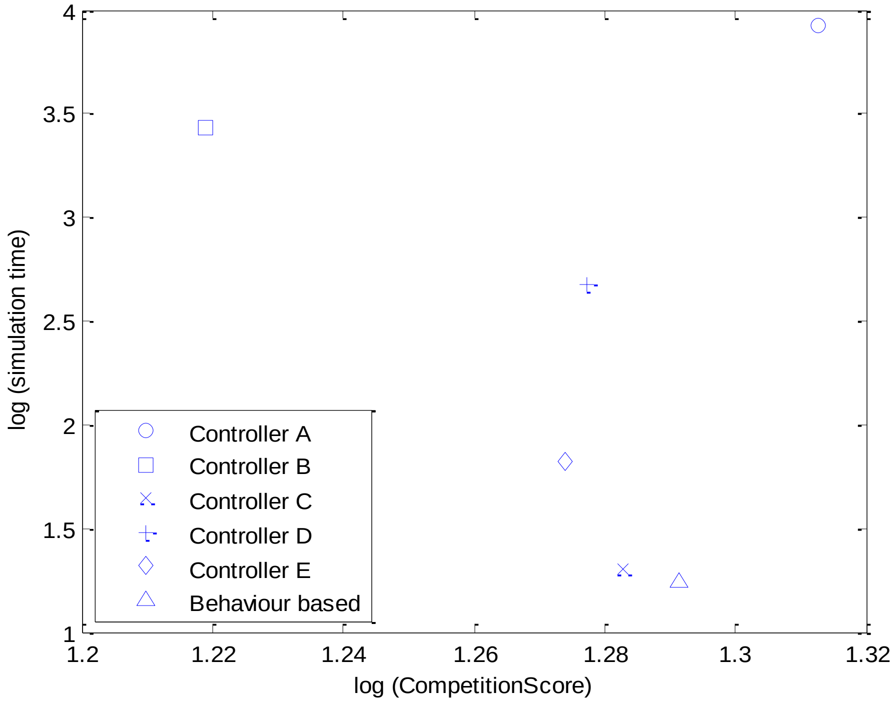
Controller	CompetitionScore	Simulation time in seconds
Controller A	20.539 (0.0416)	8450.1 (308.27)
Controller B	16.551 (0.0509)	2683.5 (217.95)
Controller C	19.176 (0.0388)	20.3 (2.31)
Controller D	18.933 (0.0662)	473.8 (73.86)
Controller E	18.797 (0.0757)	66.6 (2.675)
Behaviour-based	**19.558 (0.0536)**	**17.5 (2.42)**

In order to appreciate how the controllers performed in terms of both performance metrics simultaneously, the Pareto ranking of the controllers are considered in Figure 4.8. In a two dimensional Pareto diagram, each axes represents a performance metric. For this experiment, the two performance metrics are, simulation time on the vertical axes, and CompetitionScore on the horizontal axes. A low simulation time and a high CompetitionScore are desired. A controller is said to be dominated if there is another controller that outperforms it in both the performance metrics. Conversely, a controller is said to be non-dominated if there are no other controllers that outperforms it in both performance metrics. A controller is then given a Pareto rank defined by equation (4.3).

$$rank(i) = 1 + n_i \tag{4.3}$$

where n_i is the number of controllers dominating the individual controller i. For example, Controller E is dominated by Controller C and the behaviour-based controller, hence $n_E = 2$ and rank(E) = 3. The minimum

Pareto rank is 1. The Pareto ranks of the controllers are summarized in Table 4.9.



Low simulation time & high CompetitionScore preferred

Figure 4.8 Pareto plot of $\log_{10}$ (simulation time) against $\log_{10}$ (CompetitionScore)

Table 4.9 Pareto ranks of behaviour-based controller and top 5 controllers

Controller	Pareto rank
Controller A	1
Controller B	5
Controller C	2
Controller D	3
Controller E	3
Behaviour-based	**1**

It was observed from Table 4.9 that, according to Pareto optimality, the behaviour-based controller and Controller A obtained the highest Pareto rank of 1 amongst the 6 controllers. This meant that neither of the two controllers was completely dominant over the other controller. Controller A had obtained a higher CompetitionScore compared to the behaviour-based controller while the latter obtained a lower simulation time. In order to gain better insights to the differences between the 2 controllers, another performance indicator was required.

4.3.5.2 Round robin tournament

In CompetitionScore, all the controllers were tested on their own, and against 2 benchmark controllers. These benchmark controllers could be used during training to obtain a high CompetitionScore value. This benchmark did not test their generalization performance against unseen opponents. As such, generalization performance could be used to further distinguish the controllers. To do this, the 6 controllers were tested against one another in a round robin tournament. The scores from the round robin tournament are recorded in Table 4.10 and the t-values are also listed below each pair of scores. Each pair of controllers played 500 games against each other. The results are summarized in Table 4.11, sorted first by the number of wins, then by number of draws, then by number of losses, and finally by the total points scored. The behaviour-based controller is highlighted in bold in both tables.

From Table 4.10, it was observed that the behaviour-based controller obtained a higher mean score (significant at 0.05 level) than its opponent against all the other 5 controllers. Controller A, which had a Pareto ranking rank(A) = 1, lost against the behaviour-based controller and drew (difference in score not significant at 0.05 level) its game against Controller C. It was also noted that Controller B (Pareto ranking, rank(B) = 5) drew its game against Controller E (Pareto ranking, rank(E) = 3). From Table 4.11, the behaviour-based controller was the best performing controller with 5 wins and it also scored the highest total number of points in the tournament. This implied that the behaviour-based controller exhibit the best generalization performance amongst the 6 controllers being tested. This result also distinguished the behaviour-based controller, from the similarly Pareto ranked (rank(A) = 1) Controller A, as the better performing controller. Generalization performance is important in the context of games because of its wide array of varied customer base. If the game AI does not perform reliably well against players with different playing styles, its perceived quality will be degraded.

Table 4.10 Results for direct competition between behaviour-based controller and top 5 controllers

Controllers	A	B	C	D	E	**Behaviour-based**
A	-	17.256 (2.928) – 11.404 (2.760)	19.188 (2.424) – 19.128 (2.110)	18.234 (2.715) – 17.344 (2.258)	18.890 (2.640) – 13.736 (2.455)	**18.888 (2.417) – 19.470 (2.060)**
t-value	-	32.52	0.42	5.64	31.97	**-4.10**
B	-	-	12.912 (2.257) – 18.510 (2.102)	13.198 (2.369) – 18.142 (2.202)	11.960 (2.360) – 11.796 (2.435)	**12.694 (2.300) – 18.704 (2.145)**
t-value	-	-	-40.59	-34.18	1.08	**-42.73**
C	-	-	-	19.374 (1.923) – 18.462 (2.346)	18.522 (2.059) – 15.502 (2.281)	**19.570 (2.137) – 19.834 (1.965)**
t-value	-	-	-	6.72	21.98	**-2.03**
D	-	-	-	-	17.284 (2.342) – 12.992 (2.799)	**18.436 (2.469) – 19.570 (2.230)**
t-value	-	-	-	-	26.30	**-7.62**
E	-	-	-	-	-	**15.346 (2.321) – 18.858 (2.204)**
t-value	-	-	-	-	-	**-24.54**
Behaviour-based	-	-	-	-	-	-
t-value	-	-	-	-	-	-

Table 4.11 Consolidated results for round robin tournament of behaviour-based controller and top 5 controllers

Controller	Win / Draw / Loss	Points scored	Points against
Behaviour-based	**5 / 0 / 0**	**96.436**	**84.934**
C	3 / 1 / 1	95.104	85.898
A	3 / 1 / 1	92.456	81.082
D	2 / 0 / 3	89.668	83.368
E	0 / 1 / 4	69.372	85.514
B	0 / 1 / 4	62.168	84.408

The behaviour-based controller was able to demonstrate its generalization performance and computation efficiency through this experiment. The behaviour base controller was Pareto non-dominated in terms of CompetitionScore and simulation time, and also top ranked in the round robin competition amongst the 6 controllers. For future work, the extension of the behaviour-based controller to incorporate learning behaviours will be considered. The current behaviour-based controller is a static controller which does not learn as it plays the game. As such, good human players will be able to learn its driving patterns and develop counter strategies that can reliably win against it. In fact, I have a strategy that can reliably win against the behaviour-based controller in direct competition. In order to consistently offer a challenging game playing experience and hence upgrade the entertainment value of the game for the human player, a controller that is capable of in-game learning is desirable, but this must also work within the computational efficiency requirement of real time games.

4.4 Summary

A framework for designing computationally efficient controllers for real time games based on a hybrid evolutionary behaviour-based methodology was proposed in this chapter. The disadvantage of developing a behaviour-based controller was its requirement for hand designed components. The proposed methodology utilized genetic algorithm to complement the design of individual behavioural components. Five behaviour components were evolved using genetic algorithms. In the analysis of the evolved behaviours, it was observed that the genetic algorithm successfully exploited some collaboration between the different behaviour components which may have gone unnoticed if it was designed by hand. The best evolved controller was benchmarked against the top 5 controllers from the IEEE CEC 2007 Simulated Car Racing competition to test its generalization performance against unseen opponents. The controllers were evaluated based on their scores using the CompetitionScore benchmark, the simulation time taken, and their generalization performance in a round robin tournament against one another.

The behaviour-based controller scored the second highest in CompetitionScore but was 482 times faster than the top scoring controller. In the round robin tournament, the behaviour-based controller was able to demonstrate its better generalization capability and outperformed all the other 5 controllers. Its better computation efficiency and generalization performance makes the behaviour-based controller a more suitable candidate for implementation in real time games.

Chapter Five

5 Dynamic game difficulty scaling using adaptive game AI

Games are played by wide variety of audiences. For any given game, different individuals will play with different gaming styles and employ different strategic approaches. This often involves interacting with both the game environment and non-player characters that are controlled by the game artificial intelligence to achieve their goal. From the standpoint of a developer, it is important to design a game AI that is able to satisfy the variety of players that will interact with the game. Thus, the implementation of 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 to a specific player 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 chapter 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 rate and mutation rate are investigated for both algorithms and a general rule of thumb for the selection of these two parameters is proposed. The proposed algorithms are also demonstrated to be capable of matching its opponents in terms of mean scores and winning percentages. Both algorithms are also able to generalize well to a variety of opponent driving styles.

5.1 Introduction

Gaming is by definition an interactive experience [104]. It involves interacting with both the game environment and non-player characters (NPC) that are controlled by the game artificial intelligence (AI). In this chapter, the interaction between the player and the game AI will be examined.

High quality game AI has become an important selling point of computer games in recent years [49]. However, game players still prefer to play against human controlled opponents (via network) rather than computer controlled ones. Indeed, multi-player 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 [137]. Nevertheless, there exist situations where human game partners are unavailable such as in the absence of a viable network connection (e.g. public buses, commercial flights). In such situations, an entertaining game AI with high replay value is still desirable.

A given game is played by a wide variety of audiences who play with different gaming styles and employ 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 adding value and replayability to a game. A study with human players conducted by Hägelback & 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 [60]. Hence, the objective of this chapter is to develop an adaptive game AI that tries to entertain its opponent rather than to defeat him.

Adaptive game AI refers to a dynamic computer controlled player that adapts its game behaviour 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 behaviours 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 improve the playing experience of the game [31]. As such, 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 [172] [176]. Spronck et al introduced an adaptive algorithm that used an adaptive rulebase that can be used with current scripting game AI [144] [145]. Hunicke & Chapman controlled the game environment to make challenges easier or harder [73]. Olesen et al used rtNEAT (real time Neuro-Evolution of Augmenting Topologies) to adjust the difficult of a real time strategy game [106]. Rani et al kept the challenge at an optimal level using physiological feedback such as pulse transition time and mean temperature [123]. Bergsma & Spronck implemented ADAPTA (Allocation and Decomposition Architecture for Performing Tactical AI) that can learn and defeat static opponents in combat for a turn-based strategy game [17]. Bryant & Miikkulainen used neuroevolution to evolve a team of adaptive agents that can learn and adopt strategies in a strategy game [23]. Stanley et al used rtNEAT to allow agents in a game to adapt and improve during the game [152]. Yannakakis used evolutionary machine learning to exploit cooperative behaviours that can increase a player’s interest while playing [194]. Yannakakis & Hallam implemented an adaptive Bug-Smasher game that improved the satisfaction of children that played it [196] [199]. Thue et al used an interactive storytelling system that models a player automatically to dynamically select content to create an interactive story [168]. Riedl & Stern developed an automated story director that can adapt the plot of a story even when the player lands in an unexpected scenario [127]. Barber & Kudenko proposed an adaptive narrative engine that is able to automatically generate story events based on the interactions and decisions made by the user [14]. Quek et al used co-evolutionary 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 [120]. Tan et al experimented with adaptive rules for a minimax search tree to adapt to its opponents in Gomoku [164]. Fogel et al proposed a platform where intelligent and interactive adversarial game agents can be evolved [47]. Bellotti et al implemented an adaptive experience engine in the context of serious games [16]. Sánchez-Ruiz et al proposed an adaptive planner for turn based strategy game [132]. Bakkes et al demonstrated how domain knowledge can be gathered and adapted to new situations [10] [12]. Ponsen & Spronck used evolutionary algorithm to find new tactics to deal with opponents that were better than itself [115]. Szita et al proposed a macro learning method that can be used to generate new diverse behaviours or to adapt to its opponent [160].

Reinforcement learning [70] [156] [191] is concerned with how an agent chooses an action or sequence of action in an environment (or state) to maximize some form of long term reward. This is analogous to how a game agent acts in a game world in an effort to become the eventual winner. As such, reinforcement learning has been used to train the game AI in agent games. 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 [4] [5]. Wang et al used a reinforcement learning algorithm to improve a team of bots against its opponents in Unreal Tournament [190]. Tan used reinforcement learning in a multi-agent predator prey game to train cooperative behaviours [165].

This chapter 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 behaviour 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 chapter presents two adaptive algorithms that use ideas from reinforcement learning and evolutionary computation to play adaptively during a game session in a real time car racing simulator 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. The proposed algorithm is significant because it does not require a training phase. This will allow the human player to immediately feel the impact of adaptive behaviour from the first game played. This will also avoid the frustration a human player may feel if he is required to conduct a training phase with the game AI. This chapter also presents the first use of occurrence distribution as a measure of the performance of an adaptive game AI to play an even game.

5.2 Behaviour-based controller

The behaviour-based controller proposed in the previous chapter will be used as the basis controller [163] to develop the adaptive controller in this chapter. The various mechanisms of this controller will be briefly discussed in this section as it forms the basis for the implementation of the proposed adaptive controller.

The behaviour-based controller is inspired by behaviour-based AI [22], commonly used in the field of robotics, consisted of four independent driving behaviours that were aimed at improving the driving performance of the controller and one tactical behaviour that seek to outplay the opponent in the game. Each of the behaviour can be activated or deactivated to vary the driving behaviour of the controller. Similar behaviour selection mechanisms have also been shown to be useful in robotics [138].

Two additional tactical behaviours are introduced in this chapter to take advantage of the dynamism of such a two player competitive game. Driving behaviours ignore the existence of the opponent in the playing field, leading to inferior performance. Conversely, tactical behaviours help the controller to plan and decide which waypoint to head towards or even whether to go for any waypoint. As such, driving behaviours can be viewed as lower level operational intelligence while tactical behaviours can be viewed as higher level decision making intelligence [121]. More details about the driving and tactical behaviours can also be found in [163].

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

The following summarizes the basic behaviours inherited from the previous chapter and describes in detail the newly added tactical behaviours. It should be noted that the heading alignment behaviour has been omitted in this experiment.

Driving behaviours 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. This behaviour only acts in the forward direction.

2) Reversing

The angle to the destination was included to determine whether to drive forwards or reverse in a given situation. If the angle is within a given threshold, the speed function will be negated and the controller will reverse the car towards the destination instead.

3) Direction switching compensation

For instance, when the car is moving backwards and the destination is in the forward right direction, steering right at this point will instead orientate the car to the left, increasing the difference in heading. Instead, the controller should steer left until the reversing car comes to a halt before steering right and applying the accelerator. The same scenario applies when the car is moving forwards and the destination is behind.

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 that 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 behaviours are as follows:

5) Waypoint prediction

This is a predictive module that chooses which waypoint is more advantageous for the controller to head towards. By observing the state of the game area, this behaviour 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 towards the next waypoint instead and vise versa.

6) Time wasting

In time constrained games such as soccer, the side in possession of the ball may choose to pass the ball around in their half of the pitch and not commence any attacks. This strategy is used especially when the side in possession of the ball is in the lead and wishes to preserve their lead. In the car racing simulator, the controller may choose to stop in the proximity of the current waypoint and not drive through it if the opponent is sufficiently far away or is heading towards the next waypoint. This forces the opponent to approach the current waypoint and lose the advantage of heading towards the next waypoint.

7) Blocking

When both cars are headed towards 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 towards the current waypoint, hence increasing its chances of reaching the waypoint before its opponent does. Furthermore, if the controller also activates the reversing behaviour, it may be able to recover from a collision faster than the opponent.

5.3 Adaptive controllers

This section describes in detail the evaluation criteria used to evaluate the performance of the adaptive controllers. Two adaptive controller algorithms, the uni-chromosome adaptive controller (AUC) and the duo-chromosome adaptive controller (ADC), will also be introduced and discussed in detail.

5.3.1 Satisfying gameplay experience

A game experience is considered satisfying or entertaining when it is difficult to defeat [27]. This may be applicable to advanced players but may not necessarily apply for beginners or casual gamers. The elites, however, often only make up a small percentage of the demographics while the majority of the population is made up of low–to–medium level gamers. For this group of players, the game is most entertaining when it is challenging yet beatable [140]. Malone also pointed to challenge as one of the categories that make games fun [91]. That is, the game should neither be too easy nor too difficulty. A study with human players conducted by Hägelback & 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 [60]. In other words, in a two player competitive games, 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 [145]. However, their method required a training period of 50 encounters. The adaptive algorithms proposed in this chapter have the advantage of not requiring a training phase as the adaptation is achieved during the game session.

In the context of the car racing simulator game, there can be three possible outcomes, win, lose or draw. Therefore, in a set of n games, the player is considered most satisfied if $w = l = (n - d) / 2$ 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 chapter, 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. A high number of drawn games is deemed as more frustrating than fun.

2) $|s_1 - s_2|$ should be minimized and $\max(s_1, s_2)$ should be maximized, where s_1 and s_2 are the average individual scores of player

1 and player 2 over n games respectively. A small difference between s_1 and s_2 indicate a similar proficiency of play and a high average score indicates a competitive and fast paced game.

5.3.2 Artificial stupidity

A game is more entertaining when an opponent's mistakes are intentional but plausible [84]. 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 over-designed. 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.

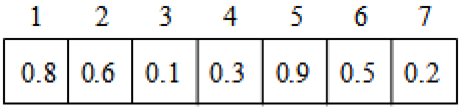
The adaptive controllers proposed in this chapter adopt a similar approach to the above analogy. An over-designed car racing simulator controller with a good set of game behaviours is first developed. In the behaviour-based AI, handicapping can be done by selectively activating or deactivating specific behaviours. The adaptive controller then estimates the ability of opposing player during the game and progressively selects a subset of behaviours to use for the remainder of the game so as to provide an engaging and satisfying game. This in turn makes the game more challenging and fun for the opposing player who now stands a chance of winning.

5.3.3 Uni-chromosome adaptive controller (AUC)

The uni-chromosome or single chromosome adaptive controller (AUC) does not need to be trained offline. The training and adaptation process occurs in real time during the game. As its name suggests, AUC stores one chromosome which encodes 7 real numbers [0,1], one for each of the 7 behaviours as shown in Figure 5.1. Each real number represents the probability of activating a behaviour module whenever a waypoint is passed. The expected behaviour set encoded by this chromosome represents a 'winning' strategy. In a sense, the chromosome models the proficiency level of the opponent by encoding a behaviour set that is expected to be 'good enough' to defeat him. It is assumed here, for simplicity, that the complement of the expected behaviour set represents a 'losing' strategy. The complement of an activation probability is calculated using equation (5.1).

$$p_i^{'} = 1 - p_i \tag{5.1}$$

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



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

The chromosome is a one-dimensional array of seven real numbers [0,1]. Each position in the chromosome corresponds to one behavioural module in the behaviour-based controller. The real number represents the probability of activating a behaviour whenever a waypoint is passed.

Figure 5.1 Representation of the chromosome used in AUC

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

```
1)      If AUC win
                for each behaviour$_i$ (i = 1 to 7)
                        if (rand() < myDist / (myDist + otherDist))
                                win$_i$ = (win$_i$ + sgn(behaviour$_i$) × l) × m;
2)      If AUC lose
                for each behaviour$_i$ (i = 1 to 7)
                        if (rand() < otherDist / (myDist + otherDist))
                                win$_i$ = (win$_i$ - sgn(behaviour$_i$) × l) × 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 i-th behaviour in the win chromosome for the next phase of the game, $behaviour_i$ is the binary state of the i-th behaviour 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 simulator, 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 its opponent, then it is easier to win this waypoint even with a weaker set of behaviour by virtue of the closer proximity. Hence, this set of behaviour should be inherited by the chromosome with lower confidence. Conversely, if the controller car wins the waypoint when it is initially further away from the destination, then that set of behaviour 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 behaviour. Whenever a waypoint is passed, the AUC checks the new game state and chooses a set of behaviour for the next phase of the game according to the values encoded in its chromosome. Each time step in a game is classified into 7 states based on the difference in score at that time step. From the perspective of the adaptive controller, the 7 states are: losing by 3 points or more (g_{-3}), losing by 2 points (g_{-2}), losing by 1 point (g_{-1}), draw (g_0), winning by 1 point (g_1), winning by 2 point (g_2) and winning by 3 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.

5.3.4 Duo-chromosome adaptive controller (ADC)

The duo-chromosome or double chromosome adaptive controller (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 2 sets of chromosomes, one winning chromosome and one losing chromosome, throughout the game. The update rules are modified as follows:

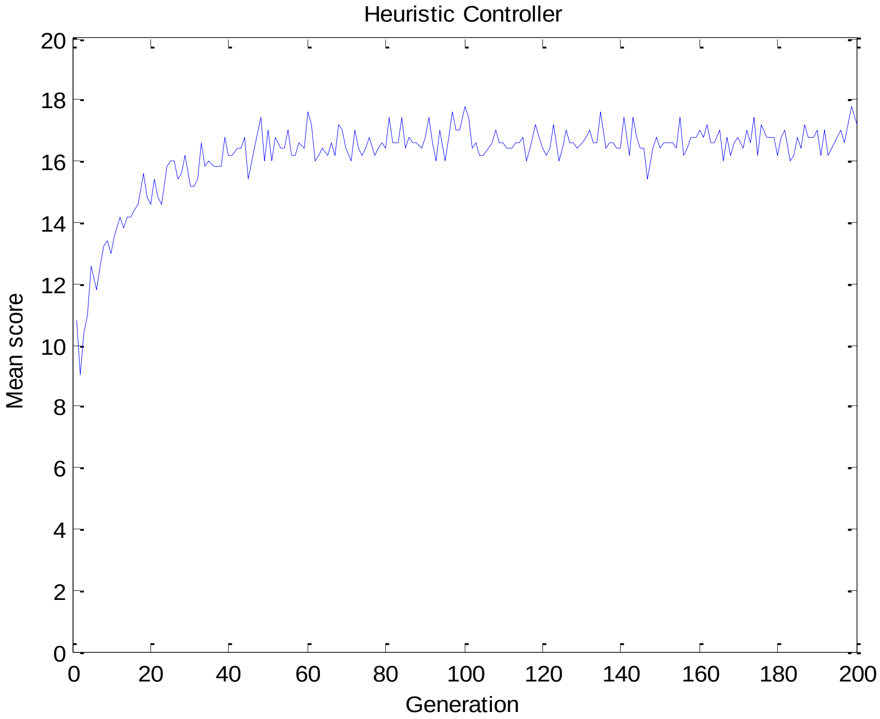
```
1)      If ADC win
                for each behaviour$_i$ (i = 1 to 7)
                        if (rand() < myDist / (myDist + otherDist))
                                win$_i$ = (win$_i$ + sgn(behaviour$_i$) × l) × m;
2)      If ADC lose
                for each behaviour$_i$ (i = 1 to 7)
                        if (rand() < otherDist / (myDist + otherDist))
                                lose$_i$ = (lose$_i$ + sgn(behaviour$_i$) × l) × 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 i-th behaviour in the win chromosome for the next phase of the game, $lose_i$ denotes the probability of activating the i-th behaviour in the lose chromosome for the next phase of the game,

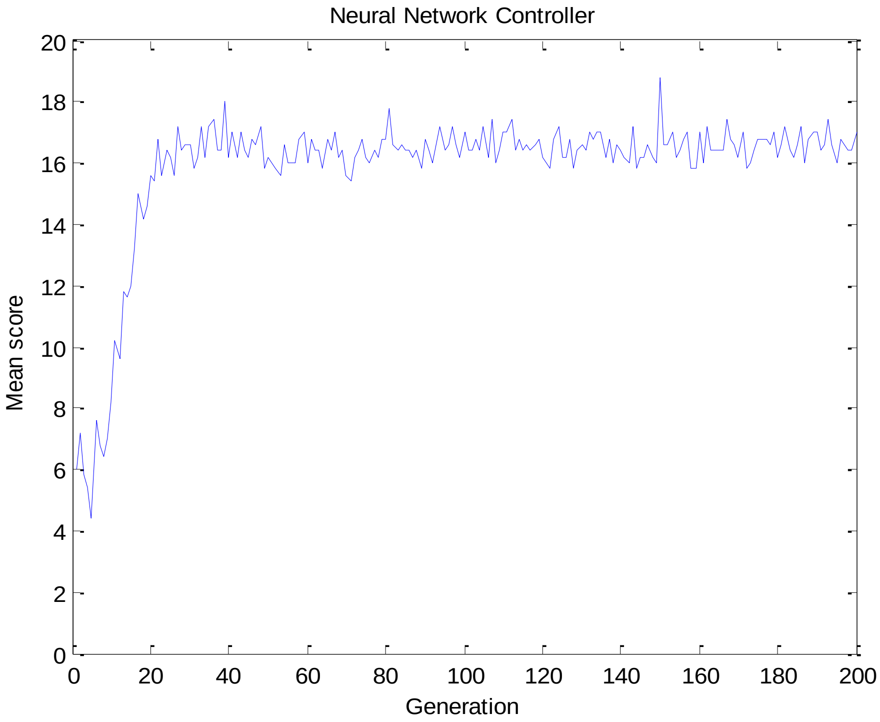
behaviour_i is the binary state of the i-th behaviour before the update, 1 for activated and -1 for deactivated, l is the learning rate, and m is the mutation rate. The mutation operator is also applied to both chromosomes after the updating process. In this controller, whenever a waypoint is passed, 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 behaviour set. If the game state is g_1, g_2 or g_3, the lose chromosome is used instead.

5.3.5 Static controllers

The adaptive controllers were tested against the following static controllers of different driving characteristics used to simulate different player with different styles of play. The purpose is to demonstrate that the adaptive algorithm is able to adapt to opponents with varying styles and competency. It should be noted that the objective of the adaptive algorithm is not to defeat its opponent. Rather, it is to play an even game.



(a)



(b)
The training fitness shown here comes from the best evolved controller from 20 independent trials of experiment.

Figure 5.2 Training fitness of (a) HC and (b) NNC

5.3.5.1 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 3 parameters, speed limit, speed limit variance and angle threshold are optimized by a plus-ES, population size of 50 and 200 generations by maximizing the number of waypoints collects against a simple hand designed heuristic controller. This controller does not have a predictive component so it ignores the existence of its opponent and always heads towards the current waypoint. The training fitness is shown in Figure 5.2 (a) is selected from the best evolved HC from 20 independent trials.

5.3.5.2 Neural network controller (NNC)

A neural network of 9 inputs, 6 hidden and 2 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. 3 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 heuristic controller. The neural network representation is capable of predictive properties but this was not seen in the best evolved candidate. However, its driving capabilities are smoother and more refined than the HC. The training fitness is shown in Figure 5.2 (b) is selected from the best evolved NNC from 20 independent trials.

5.3.5.3 Reverse enabled controller (RC)

The reverse enabled controller (RC) is a simplification of the behaviour-based controller described earlier. The hyperbolic tangent speed regulator was deactivated and replaced with a hard speed limit of 5. Only the reversing and direction switching compensation behaviour was activated while all other driving and tactical behaviours were deactivated. This controller ignores the opponent in the two player game but makes good use of its ability to drive both forwards and backwards to earn points, and recovers quickly from collisions.

5.3.5.4 Predictive slow controller (PSC)

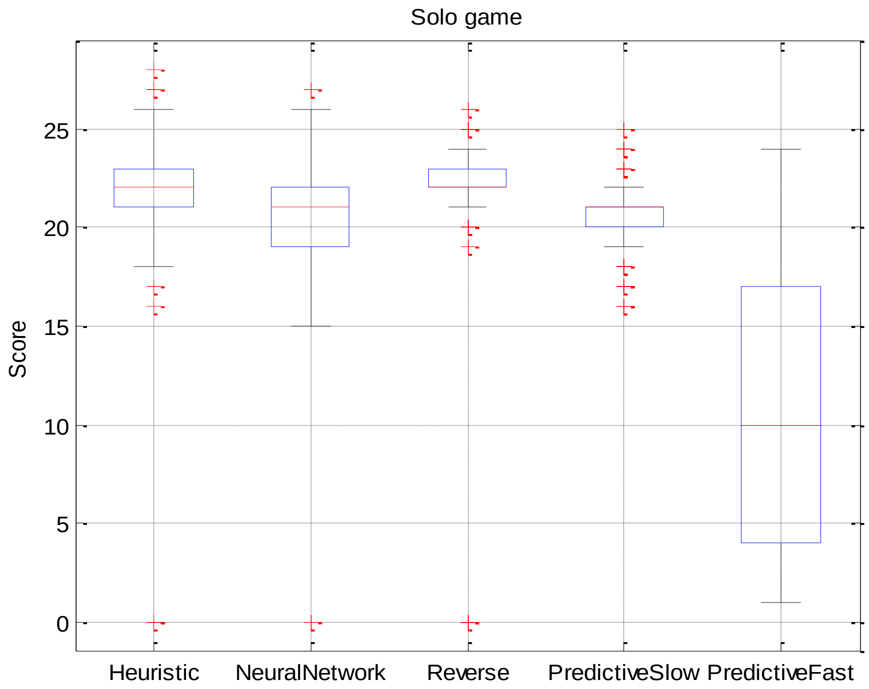
The predictive slow controller (PSC) is an extension of the HC with the addition of the waypoint prediction component from the set of tactical behaviours. The speed limit is set to 5 to simulate a relatively slower moving car compared to the HC.

5.3.5.5 Predictive fast controller (PFC)

The predictive fast controller (PFC) is a variation of the PSC but with the speed limit set to 8 instead, a car faster than the HC. 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 sufficient close to its destination.

5.3.5.6 Solo game

The results of the solo run of the static controllers presented in Figure 5.3 gives an indication of the driving capabilities of the controller. Since there were no opponent vehicles in this mode, PSC and PFC reduced to a variant of the HC but with different speed limits in the solo case.



An outlier at zero score indicates that the controller is stuck in orbit at the very first waypoint.
The results were obtained from n = 5000 games.

Figure 5.3 Comparative results of static controllers in solo games

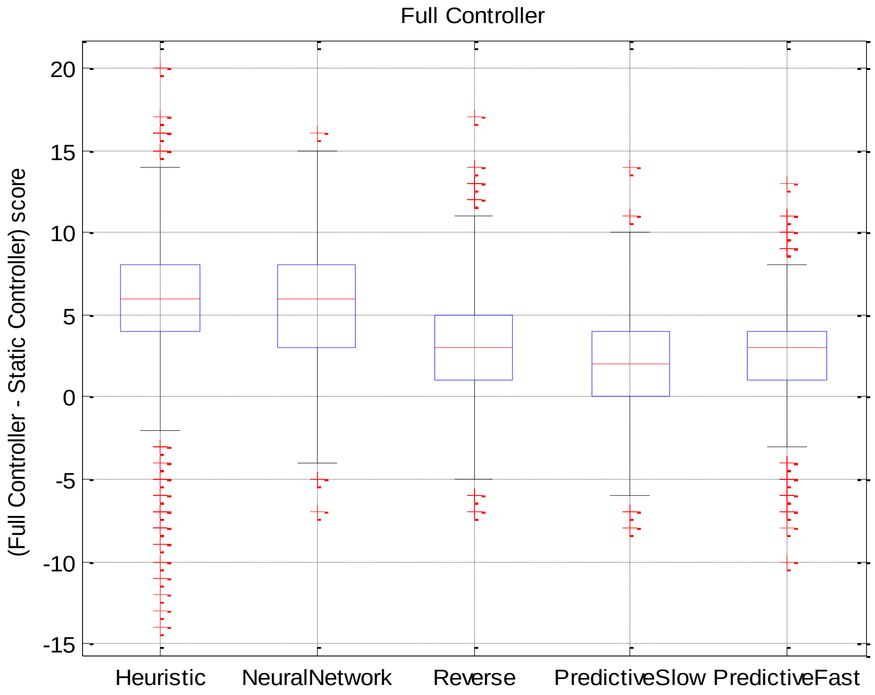
It was observed that the controllers HC, NNC, RC and PSC exhibit similar driving capabilities with controller NNC being slightly less consistent. Controller PFC was the worst performing controller with a much lower average score and was very inconsistent with a large standard deviation. The reason was that its high speed limits often resulted in skidding during a turn and hence it often overshot a waypoint, wasting valuable time. However, controller PFC did not report any games with zero score outlier, unlike controllers HC, NNC and RC. A zero score outlier indicated that the controller was stuck in orbit at the very first waypoint. This showed that a low speed limit (PSC) and a stopping mechanism (PFC) are effective in preventing orbiting.

5.4 Results and analysis

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.

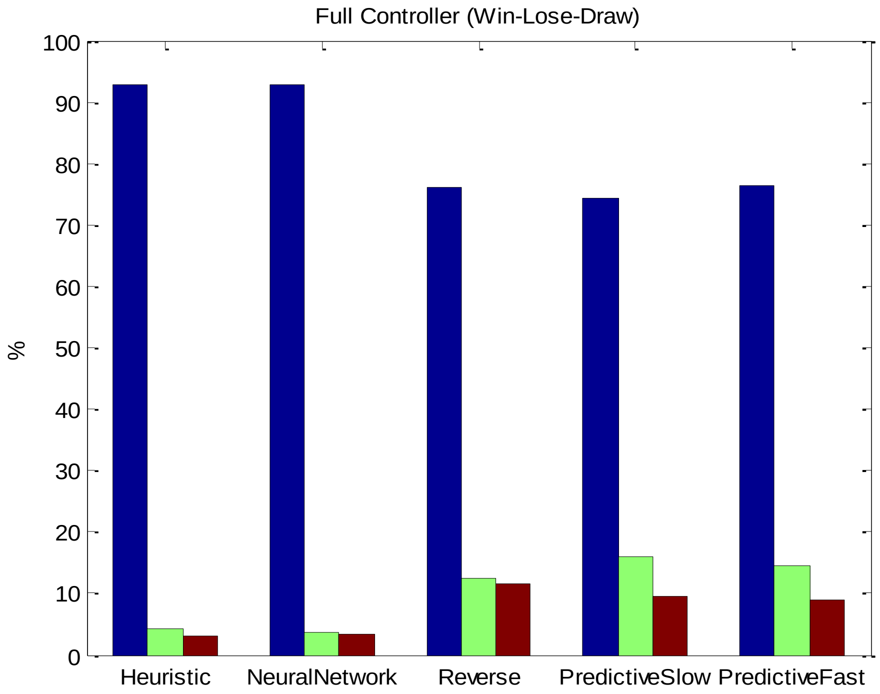
5.4.1 Fully activated behaviours

The first step of the experiment was to establish the playing proficiency of the basis behaviour-based controller to be used in the adaptive algorithms. There was a need to verify whether or not the basis behaviour-based controller was indeed an over-design and hence possessed the potential to play even games against its opponents. The most competent controller was one with all behaviours permanently activated and represented the adaptive controller playing at full strength throughout each game. The full controller (FC) was played against each of the five static controllers. 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 Figure 5.4. A positive score difference indicated that the FC won a particular game while a negative score difference indicated 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 of its opponents are presented in Figure 5.5. Each collection of three connected bars represents win, lose and draw percentages from left to right.



The results are shown in terms of score differences between the FC and its opponent. A positive score difference indicates that the FC won; while a negative score difference indicates that the opponent won.

Figure 5.4 Boxplot of the results from playing the FC against the five static controllers



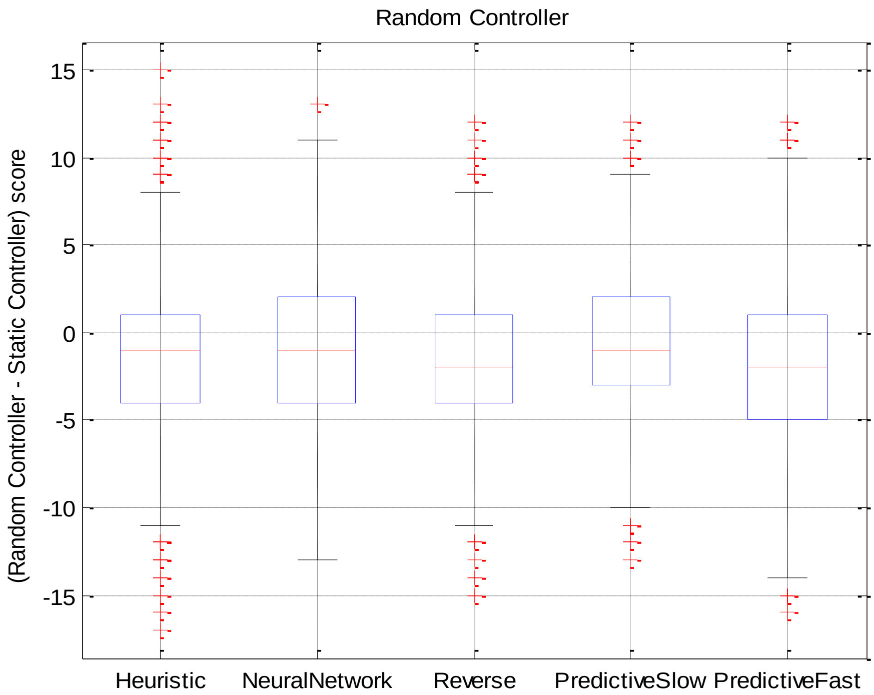
For each static controller, the three histogram bars from left to right (blue, green and red) represent the percentage of games that the FC won, lost and drew respectively.

Figure 5.5 Histogram of the results from playing the FC against the five static controllers

It was observed that the FC was a very competent controller with positive median score differences against all its opponents. The lower quartile score differences were positive against four of its opponents and zero against only the PSC. This could also be observed in Figure 5.5 where the FC obtained the lowest winning percentage of 74.42% against the PSC. However, it was clear that the FC was a very competent player against the static controllers. This made 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 it in winning and losing percentages.

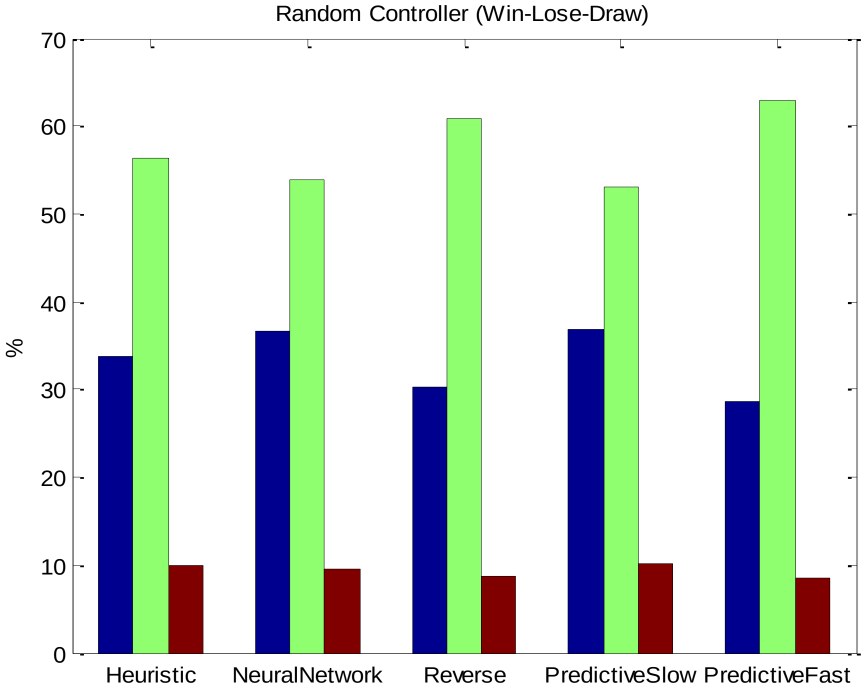
5.4.2 Randomly activated behaviours

Besides having a suitable candidate for adaptation, it was also important to know whether or not adaptation was a necessity. A simple random algorithm was used in this experiment to demonstrate the need for guided learning in an adaptive algorithm. The random controller (RDC) operated by randomly picking a new set of behaviour to use every time a waypoint was passed. The RDC was played against each of the five static controllers 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 Figure 5.6. A positive score difference indicated 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 Figure 5.7 were from the perspective of the RDC. Each group of three connected bars represents win, lose and draw percentages from left to right.



The results are shown in terms of score differences between the RDC and its opponent. A positive score difference indicates that the RDC won; while a negative score difference indicates that the opponent won.

Figure 5.6 Boxplot of the results from playing the RDC against the five static controllers



For each static controller, the three histogram bars from left to right (blue, green and red) represent the percentage of games that the RDC won, lost and drew respectively.

Figure 5.7 Histogram of the results from playing the RDC against the five static controllers

It was observed from Figure 5.6 that the median of the score differences were all negative. That is, the score of the RDC was lower than that of its opponent. This could be confirmed in Figure 5.7 by observing that the RDC had a higher losing percentage than winning percentage against all five opponents. This implied that a competent player playing with random behaviours was unable to consistently win against players of lower competency. Hence, the random use of effective behaviours did not equate to a good player. Furthermore, the choice of what behaviours to use to match an opponent in a game needs to be guided.

5.4.3 Analysis of AUC

The AUC stores one chromosome which encodes 7 real numbers in the range [0,1], one for each of the 7 behaviours. Each value represents the probability of activating a behaviour for use. The expected behaviour set encoded by this chromosome represents a 'winning' strategy. Whenever a waypoint is passed in the game, the chromosome is updated by the rules described earlier in section 5.3.3 and a new set of behaviours will be generated for use until the next waypoint is triggered. The proposed algorithm introduced 2 variables, the learning rate and the mutation rate. The effects of varying these variables will be discussed in this section.

5.4.3.1 Effects of varying learning rate

In this experiment, the mutation rate was set to zero and the learning rate was varied from 0.1 to 1.0 in steps of 0.1. The results of the mean scores, standard deviation and winning percentages over n = 5000 games are summarized in Table 5.1.

Table 5.1 Comparative results for AUC versus static controllers for varying learning rate and fixed mutation rate

For each pair of results, the left column represents the AUC and the right represents the static controller in that column. Winning percentages do not sum to 100%, the remainder are drawn games. The best results are bolded for each static controller.

AUC Mutation rate = 0											
Learning rate		Heuristic		Neural network		Reverse		Predictive slow		Predictive fast	
0.1	Mean	**14.10**	**14.10**	**14.00**	**14.00**	**14.46**	**14.44**	**14.72**	**14.71**	14.55	14.54
	Std	2.37	2.67	2.45	2.55	2.29	2.88	2.62	3.12	2.58	2.89
	Win (%)	**45.24**	**45.24**	44.58	45.26	**44.98**	**45.64**	**44.66**	**44.66**	43.64	44.48
0.2	Mean	14.10	14.08	13.92	13.91	14.59	14.43	14.90	14.78	**14.72**	**14.72**
	Std	2.39	2.66	2.48	2.58	2.26	2.80	2.49	3.04	2.46	2.81
	Win (%)	45.52	45.20	44.72	45.34	44.14	46.42	43.98	46.22	44.50	45.12
0.3	Mean	14.15	14.13	14.07	14.06	14.68	14.08	15.04	14.66	14.82	14.83
	Std	2.37	2.56	2.46	2.55	2.24	2.67	2.48	2.89	2.42	2.70
	Win (%)	45.50	45.08	44.76	45.46	39.72	51.18	41.06	48.60	44.16	44.68
0.4	Mean	14.22	13.97	14.09	14.08	14.70	14.11	15.06	14.57	14.84	14.83
	Std	2.40	2.53	2.45	2.46	2.23	2.68	2.44	2.82	2.32	2.63
	Win (%)	42.50	48.52	**45.22**	**45.82**	39.92	50.90	39.44	50.74	**44.42**	**44.84**
0.5	Mean	14.20	14.02	14.22	14.11	14.75	14.00	15.05	14.60	14.92	14.85
	Std	2.39	2.48	2.39	2.37	2.24	2.68	2.45	2.78	2.31	2.60
	Win (%)	43.80	46.36	44.58	45.58	38.56	52.66	39.66	50.30	43.28	45.96
0.6	Mean	14.24	13.89	14.25	14.03	14.80	13.93	15.11	14.50	14.98	14.89
	Std	2.39	2.47	2.47	2.41	2.22	2.65	2.44	2.76	2.22	2.51
	Win (%)	42.88	47.56	42.92	47.36	36.82	53.50	38.50	51.36	43.32	45.56
0.7	Mean	14.26	13.82	14.15	14.00	14.81	13.94	15.12	14.38	14.99	14.77
	Std	2.36	2.40	2.39	2.35	2.23	2.61	2.50	2.80	2.26	2.53
	Win (%)	41.40	49.42	44.64	46.26	36.72	53.92	37.90	53.08	42.90	46.96
0.8	Mean	14.27	13.81	14.21	13.97	14.81	13.91	15.11	14.33	14.95	14.75
	Std	2.34	2.41	2.44	2.36	2.20	2.58	2.50	2.79	2.25	2.51
	Win (%)	41.44	49.32	42.52	47.52	36.94	53.94	36.84	52.94	41.98	46.88
0.9	Mean	14.30	13.79	14.26	13.98	14.86	13.88	15.17	14.38	14.94	14.71
	Std	2.40	2.42	2.42	2.34	2.16	2.51	2.47	2.80	2.31	2.48
	Win (%)	40.10	50.82	42.68	47.54	35.40	54.18	36.96	52.90	41.96	47.46
1.0	Mean	14.22	13.85	14.22	13.92	14.81	13.78	15.04	14.32	14.94	14.69
	Std	2.35	2.38	2.40	2.34	2.23	2.58	2.51	2.79	2.32	2.52
	Win (%)	42.50	47.96	42.56	47.72	35.74	54.86	37.80	52.82	41.82	47.92

The results were evaluated based on the two criteria described in section 5.3.1. The difference in winning percentage |w – l| should be minimal and the number of draws d should also be minimized. A high number of drawn games was deemed as more frustrating than fun. The difference between the mean scores $|s_1 - s_2|$ should be minimal and the higher of the two scores $\max(s_1, s_2)$ should be maximal. A high average score indicated a competitive and fast paced game that was deemed to provide more satisfaction for the player. The best results based on these criteria are highlighted in bold in Table 5.1.

It was observed from Table 5.1 that the general trend of increasing learning rate was an increase in mean score differences and also an increase in winning percentage difference. This was 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 behaviours that were unable to adapt and track its opponent's progress during the game. At low learning rates, the score differences and winning percentages were smaller and the AUC was able to match its opponent in both criteria. A learning rate of 0.1 obtained the best result for 7 out of 10 evaluations (2 evaluation criteria for each of 5 static controllers). It was also the dominant learning rate for 3 out of 5 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 were considered within acceptable range. Therefore, a learning rate of 0.1 was chosen as a good general rule of thumb that could be used in situations where opponents were varied and unknown. This value of learning rate will also be used as default value in the experiment of varying mutation rate in the next section.

It was also worth noting that in the lower half of Table 5.1 (i.e. l > 0.5), the mean score of the adaptive controller was higher than that of the static controllers but the winning percentages of the AUC is lower than that of the static controllers. This was likely caused by the AUC losing frequently by small margins but winning by large margins. This exemplified that higher mean scores did not directly imply higher winning percentages.

5.4.3.2 Effects of varying mutation rate

In this experiment, the learning rate was set to 0.1 and the mutation rate was varied from 0.1 to 1.0 in steps of 0.1. The mutation rate controlled the size of the standard deviation in a Gaussian perturbation of zero mean. The mutations were applied independently to each chromosome value after the learning rate was applied. The results of the mean scores, standard deviation and winning percentages are summarized in Table 5.2.

Table 5.2 Comparative results for AUC versus static controllers for fixed learning rate and varying mutation rate

For each pair of results, the left column represents the AUC and the right represents the static controller in that column. Winning percentages do not sum to 100%, the remainder are drawn games. The best results are bolded for each static controller.

AUC Learning rate = 0.1 Mutation rate		Heuristic		Neural network		Reverse		Predictive slow		Predictive fast	
0.1	Mean	**14.12**	**14.06**	**14.05**	**14.06**	**14.64**	**13.98**	**15.01**	**14.54**	**14.81**	**14.79**
	Std	2.37	2.60	2.44	2.52	2.22	2.78	2.52	3.01	2.41	2.70
	Win (%)	44.08	46.68	44.74	45.42	**39.50**	**51.52**	**40.70**	**49.48**	40.90	47.52
0.2	Mean	14.10	13.92	14.13	14.04	14.72	13.84	15.05	14.40	14.94	14.88
	Std	2.37	2.50	2.42	2.41	2.25	2.64	2.52	2.83	2.27	2.53
	Win (%)	**44.08**	**46.22**	**45.02**	**45.46**	36.34	54.06	38.20	51.90	**44.26**	**44.60**
0.3	Mean	14.11	13.88	14.18	13.95	14.74	13.80	15.11	14.33	14.88	14.85
	Std	2.34	2.38	2.36	2.30	2.16	2.54	2.50	2.83	2.34	2.53
	Win (%)	42.58	47.38	43.48	46.88	36.14	53.74	37.18	53.44	43.96	45.26
0.4	Mean	14.18	13.82	14.17	13.96	14.70	13.85	15.13	14.28	14.95	14.87
	Std	2.27	2.31	2.38	2.27	2.16	2.50	2.48	2.75	2.24	2.46
	Win (%)	41.44	48.62	43.02	47.04	37.74	53.28	35.64	54.38	44.26	45.02
0.5	Mean	14.19	13.74	14.15	13.92	14.79	13.72	15.15	14.27	14.95	14.87
	Std	2.31	2.34	2.34	2.25	2.20	2.54	2.42	2.69	2.32	2.46
	Win (%)	41.24	48.56	42.76	47.04	33.92	55.70	36.14	53.86	43.02	45.16
0.6	Mean	14.26	13.71	14.24	13.87	14.76	13.71	15.09	14.28	14.91	14.82
	Std	2.32	2.30	2.40	2.22	2.14	2.42	2.43	2.64	2.25	2.43
	Win (%)	40.26	49.58	41.26	48.80	34.44	54.84	36.38	53.44	43.26	46.22
0.7	Mean	14.26	13.71	14.20	13.85	14.78	13.69	15.15	14.19	14.97	14.75
	Std	2.30	2.32	2.38	2.27	2.15	2.43	2.48	2.71	2.25	2.38
	Win (%)	40.36	49.98	41.72	47.90	34.23	56.22	35.08	54.86	43.20	46.30
0.8	Mean	14.24	13.72	14.25	13.79	14.82	13.60	15.06	14.14	14.98	14.72
	Std	2.36	2.32	2.37	2.22	2.18	2.53	2.57	2.76	2.25	2.40
	Win (%)	40.74	49.14	40.26	49.54	33.62	56.66	35.40	54.40	41.20	47.34
0.9	Mean	14.29	13.65	14.18	13.82	14.84	13.64	15.14	14.12	14.99	14.71
	Std	2.34	2.22	2.38	2.21	2.17	2.46	2.46	2.63	2.25	2.36
	Win (%)	39.08	50.76	41.20	48.82	34.34	56.24	34.38	54.80	41.02	47.32
1.0	Mean	14.26	13.68	14.27	13.83	14.88	13.59	15.10	14.06	14.98	14.74
	Std	2.34	2.25	2.37	2.21	2.18	2.47	2.54	2.74	2.26	2.43
	Win (%)	38.70	51.00	41.60	48.82	33.00	58.46	34.40	55.08	40.66	47.64

It was observed from Table 5.2 that higher mutation rates was more likely to produce larger differences in mean score and winning percentage. Similar to the case of high learning rate, high mutation rate produced large fluctuations in the chromosome values, making the AUC overcompensate in its behaviours. This was analogous to noise being amplified by the differential component of a PID (proportional–integral–derivative) controller. The best performing mutation rate was 0.1 with 7 out of 10 best evaluations. 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, were worse (i.e. larger differences in mean score and winning percentage) compared to those from m = 0. This implied that the additional mutation operation might have introduced unnecessary divergence to the chromosome values, leading to poorer results. Therefore, in general, the mutation rate should be 0 for the AUC.

5.4.4 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 7 real number in the range [0,1]. One chromosome represents a 'winning' behaviour set while the other represents a 'losing' behaviour set. Each chromosome is updated independently when a waypoint is passed. However, the same learning rate and mutation rate is applied to both chromosomes.

5.4.4.1 Effects of varying learning rate

In this experiment, the mutation rate was first set to zero and the learning rate was varied from 0.1 to 1.0 in steps of 0.1. The same learning rate was applied to both chromosomes of the ADC. The results of the mean scores, standard deviation and winning percentages are summarized in Table 5.3. As is the case in Table 5.1, the best results are highlighted in bold.

Table 5.3 Comparative results for ADC versus static controllers for varying learning rate and fixed mutation rate

For each pair of results, the left column represents the ADC and the right represents the static controller in that column. Winning percentages do not sum to 100%, the remainder are drawn games. The best results are bolded for each static controller.

ADC Mutation rate = 0											
Learning rate		Heuristic		Neural network		Reverse		Predictive slow		Predictive fast	
0.1	Mean	8.16	10.81	**12.16**	**12.16**	**14.31**	**14.31**	**10.23**	**10.23**	**11.78**	**11.78**
	Std	2.57	2.55	2.85	2.98	2.57	2.80	3.36	3.09	3.29	3.23
	Win (%)	73.00	18.48	44.74	43.10	43.38	43.80	43.98	41.48	**41.16**	**40.22**
0.2	Mean	11.12	13.10	13.99	13.25	14.24	13.60	15.22	14.54	9.29	10.01
	Std	2.00	1.95	2.34	1.80	2.20	2.33	2.38	2.31	2.23	2.10
	Win (%)	65.76	24.08	39.40	51.96	38.80	51.68	37.06	52.36	54.72	32.98
0.3	Mean	13.87	12.34	7.93	10.87	9.93	11.28	9.75	9.76	8.97	9.99
	Std	2.27	1.98	2.54	2.47	4.08	4.53	2.55	2.32	2.12	2.01
	Win (%)	**29.70**	**60.42**	76.28	16.38	58.18	27.16	44.68	41.78	57.40	29.52
0.4	Mean	8.81	13.35	12.42	13.45	12.14	16.19	9.43	9.46	14.67	15.98
	Std	2.58	2.90	2.30	1.93	1.91	2.17	2.40	2.16	2.39	2.27
	Win (%)	88.94	5.84	55.50	34.98	82.44	10.98	43.84	41.60	59.34	29.62
0.5	Mean	14.19	12.22	12.05	12.05	8.47	9.64	8.74	9.25	8.94	9.94
	Std	2.15	2.31	2.53	2.29	4.46	5.13	2.49	2.24	2.18	1.99
	Win (%)	26.70	63.98	**44.86**	**44.38**	55.64	27.46	51.72	35.24	57.78	30.14
0.6	Mean	13.13	11.04	14.29	11.97	12.13	12.13	8.92	9.44	14.42	13.44
	Std	2.20	1.92	2.26	1.81	4.36	4.37	2.45	2.17	2.30	2.61
	Win (%)	24.18	66.76	21.70	69.18	**42.48**	**42.38**	51.78	35.40	34.12	55.94
0.7	Mean	8.11	10.86	9.92	13.33	14.27	13.44	14.71	18.74	9.35	12.55
	Std	2.57	2.54	2.33	2.24	2.25	2.38	2.65	2.99	2.32	2.24
	Win (%)	74.10	17.72	79.84	13.36	37.06	53.24	85.06	9.28	81.60	11.00
0.8	Mean	**13.08**	**11.17**	9.40	13.72	11.45	14.78	9.43	9.44	9.37	12.51
	Std	2.20	1.91	2.28	2.28	1.74	1.91	2.34	2.13	2.28	2.27
	Win (%)	25.72	65.00	87.68	7.32	78.72	13.82	43.74	41.96	81.04	10.68
0.9	Mean	11.50	16.66	9.95	13.41	14.25	20.62	15.24	13.38	10.74	13.50
	Std	2.00	1.91	2.37	2.21	2.04	1.97	2.18	2.46	2.29	2.19
	Win (%)	93.00	3.36	79.68	13.64	96.04	2.02	27.04	63.94	79.98	11.64
1.0	Mean	10.24	13.07	13.97	13.26	12.63	13.58	13.40	13.39	9.29	9.98
	Std	2.44	2.10	2.37	1.80	2.16	2.54	2.87	3.05	2.14	2.04
	Win (%)	76.26	16.44	38.40	51.46	53.66	36.02	**43.58**	**44.24**	53.72	33.06

It was observed in Table 5.3 there were no trends with varying the learning rate. This was likely caused by the reduction of the frequency of update opportunities for each chromosome. The average number of updates during each game was the sum of the mean scores of the two players. With the ADC, only one chromosome was updated whenever a waypoint was passed depending on whether the controller won or lost the point. This meant that, on average, each chromosome in the ADC was 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, a learning rate of 0.1 obtained the best result in 5 out of 10 evaluations. Therefore, l = 0.1 will be used as the default value in the experiment of varying mutation rate in the next section.

5.4.4.2 Effects of varying mutation rate

In this experiment, the learning rate was set to 0.1 and the mutation rate was varied from 0.1 to 1.0 in steps of 0.1. The mutation rate controlled the size of the standard deviation in a Gaussian perturbation of zero mean. The mutations were applied independently to each chromosome value after the learning rate was applied. The same mutation rate was applied to both chromosomes of the ADC. The results of the mean scores, standard deviation and winning percentages are summarized in Table 5.4.

Table 5.4 Comparative results for ADC versus static controllers for fixed learning rate and varying mutation rate

For each pair of results, the left column represents the ADC and the right represents the static controller in that column. Winning percentages do not sum to 100%, the remainder are drawn games. The best results are bolded for each static controller.

ADC Learning rate = 0.1 Mutation rate		Heuristic		Neural network		Reverse		Predictive slow		Predictive fast	
0.1	Mean	**13.89**	**13.88**	**13.97**	**13.96**	**14.43**	**14.38**	**14.93**	**14.71**	14.27	14.26
	Std	2.45	2.86	2.52	2.84	2.28	3.03	2.51	3.20	2.74	3.13
	Win (%)	**45.82**	**45.22**	**45.58**	**45.22**	**45.28**	**46.26**	**44.06**	**46.68**	**44.40**	**44.66**
0.2	Mean	14.15	13.91	13.95	13.91	14.68	13.91	15.01	14.46	**14.79**	**14.79**
	Std	2.32	2.53	2.47	2.55	2.17	2.71	2.51	2.92	2.41	2.67
	Win (%)	42.68	47.04	45.42	45.02	37.96	52.10	39.42	51.18	44.18	44.90
0.3	Mean	14.17	13.89	14.07	14.01	14.71	13.90	15.01	14.44	14.90	14.89
	Std	2.30	2.43	2.41	2.38	2.20	2.61	2.42	2.79	2.34	2.54
	Win (%)	42.40	47.26	44.20	45.96	37.90	52.68	38.80	50.22	44.80	44.66
0.4	Mean	14.17	13.82	14.09	14.04	14.68	13.81	15.02	14.32	14.86	14.82
	Std	2.27	2.33	2.38	2.31	2.17	2.57	2.44	2.79	2.38	2.52
	Win (%)	41.62	48.20	44.62	45.52	36.46	53.92	37.50	51.94	43.82	45.36
0.5	Mean	14.19	13.80	14.14	13.92	14.66	13.78	15.06	14.29	15.02	14.94
	Std	2.34	2.34	2.39	2.31	2.19	2.54	2.51	2.76	2.30	2.40
	Win (%)	42.24	48.70	43.22	47.66	35.86	53.80	36.56	52.88	43.66	44.94
0.6	Mean	14.17	13.82	14.24	13.91	14.75	13.75	15.11	14.22	14.91	14.80
	Std	2.31	2.30	2.34	2.28	2.17	2.48	2.42	2.69	2.33	2.47
	Win (%)	41.74	48.30	42.54	48.00	34.74	55.56	35.32	53.66	43.30	45.88
0.7	Mean	14.26	13.66	14.16	13.93	14.71	13.77	15.14	14.20	14.97	14.88
	Std	2.34	2.30	2.33	2.24	2.17	2.47	2.49	2.67	2.23	2.43
	Win (%)	38.98	51.16	43.06	46.68	36.16	54.22	35.66	55.08	43.12	45.38
0.8	Mean	14.20	13.75	14.19	13.89	14.73	13.69	15.10	14.21	14.97	14.89
	Std	2.31	2.28	2.40	2.23	2.21	2.49	2.50	2.71	2.20	2.34
	Win (%)	40.94	48.96	43.08	46.70	34.70	55.62	36.76	53.46	42.94	45.90
0.9	Mean	14.24	13.67	14.23	13.86	14.86	13.61	15.12	14.09	14.98	14.77
	Std	2.33	2.29	2.36	2.21	2.14	2.42	2.46	2.68	2.27	2.41
	Win (%)	40.28	49.68	41.68	48.46	32.16	57.98	34.82	54.92	41.64	46.34
1.0	Mean	14.23	13.66	14.20	13.93	14.80	13.61	15.20	14.15	14.92	14.76
	Std	2.32	2.25	2.34	2.23	2.18	2.48	2.38	2.63	2.25	2.37
	Win (%)	39.78	49.46	42.24	47.44	33.52	57.24	33.66	56.06	42.54	46.06

It was observed in Table 5.4 that the mutation value of 0.1 obtained the best result in 9 out of 10 evaluations. The only exception was for the case of mean score difference against the PFC. However, their mean scores only differed by 0.01 and could be considered within acceptable range. The results of the ADC improved greatly due to the introduction of the mutation operation as the mutation operation offered more opportunities for the chromosome values to adapt compared to using only the learning rate operation. The mean scores and winning percentages were similar to those of the AUC. Additionally, the ADC (l = 0.1, m = 0.1, 9 of 10 evaluations) was able to produce more consistent results compared to the AUC (l = 0.1, m = 0, 7 of 10 evaluations) in response to varied and unknown opponents. The disadvantage was that the ADC requires more memory and more computation.

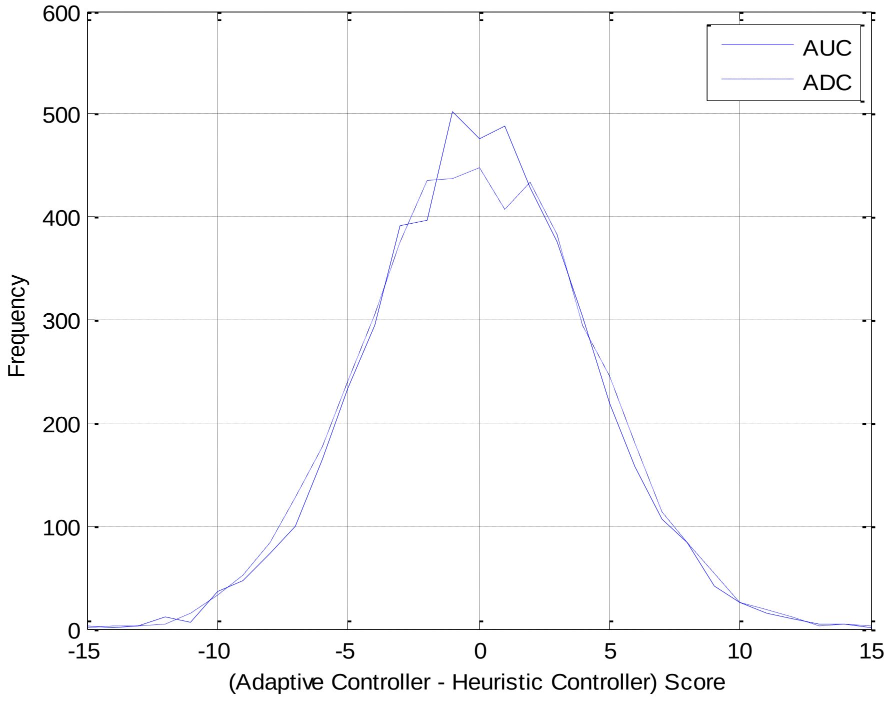
5.4.5 Score difference distribution

Both the AUC and ADC, using the optimized parameters, were demonstrated to be effective in matching its 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.

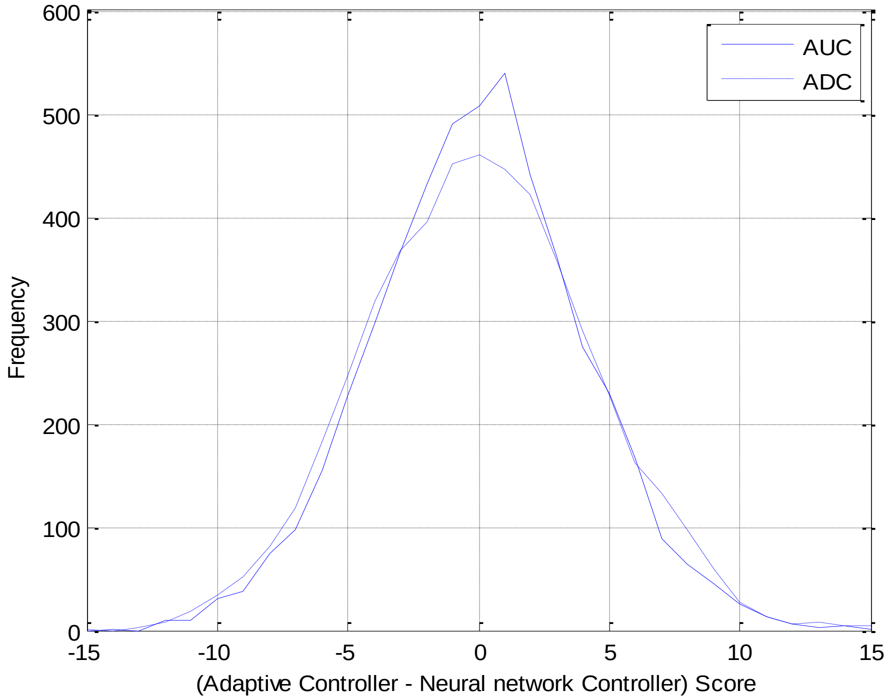
5.4.5.1 Distribution of score difference

The significance of analyzing the distribution of score differences is to investigate the effects of the adaptive controllers 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 car racing simulator game, this can be interpreted as a small score difference between the two competing players. From real world user experience, a game was subconsciously considered won or lost by a player when a score difference of more than 5 was observed during the game. The typical score of a player for one game was around 13 to 15 points. Therefore, if the end game score difference is 4 or less, it is said that the player is entertained during the game.

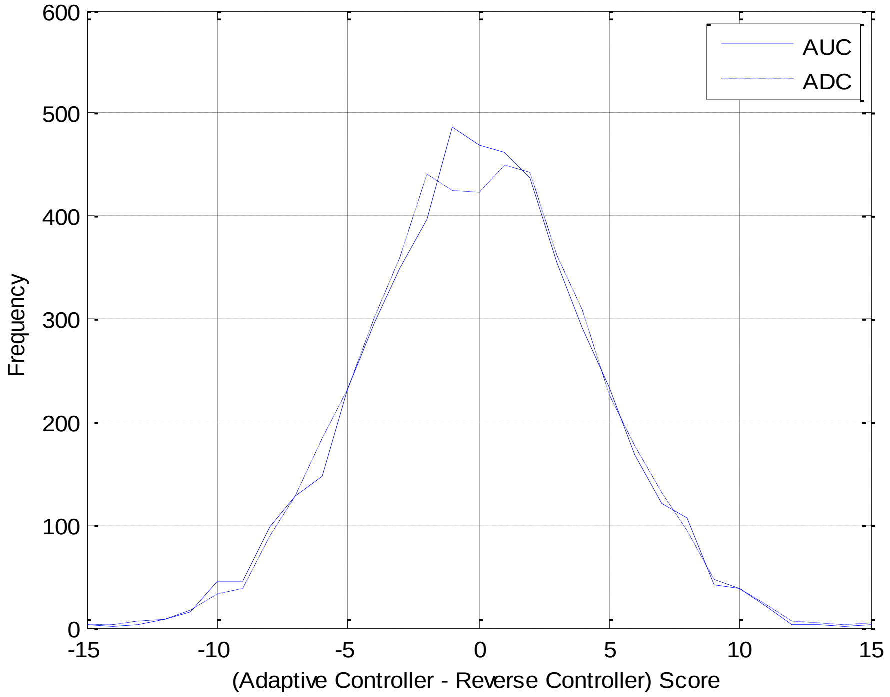
The histograms of the score difference are presented in Figure 5.8. The boxplot of the score difference of the AUC and the ADC with optimized parameters against the five static opponents are presented in Figure 5.9 and Figure 5.10 respectively. The number of game results that fall within a specific score difference is summarized in Table 5.5 as a percentage of total games played.



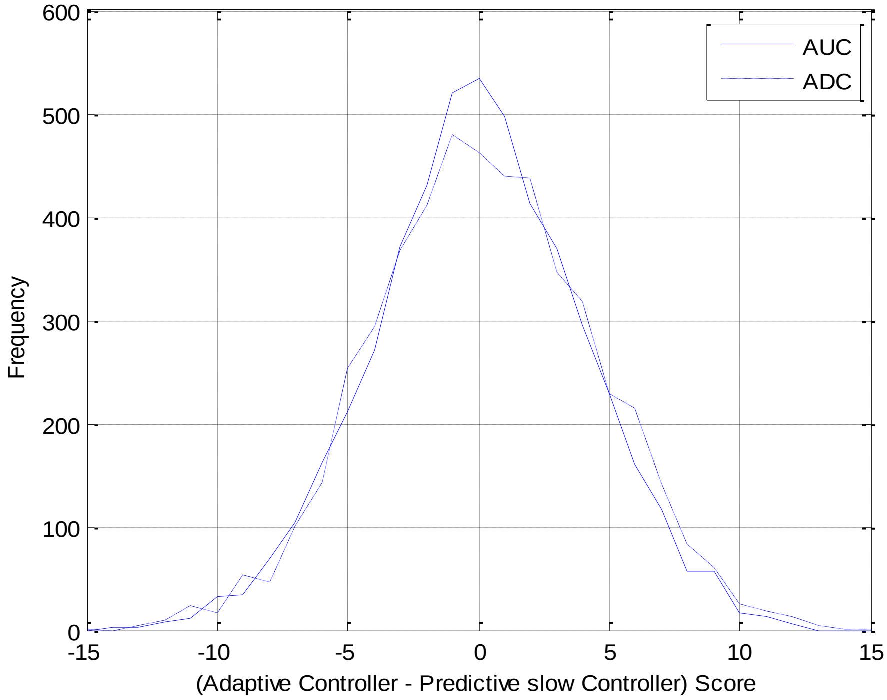
(a)


600
500
400
300
200
100
0
Frequency
-15
-10
-5
0
5
10
15
(Adaptive Controller - Neural network Controller) Score
AUC
ADC

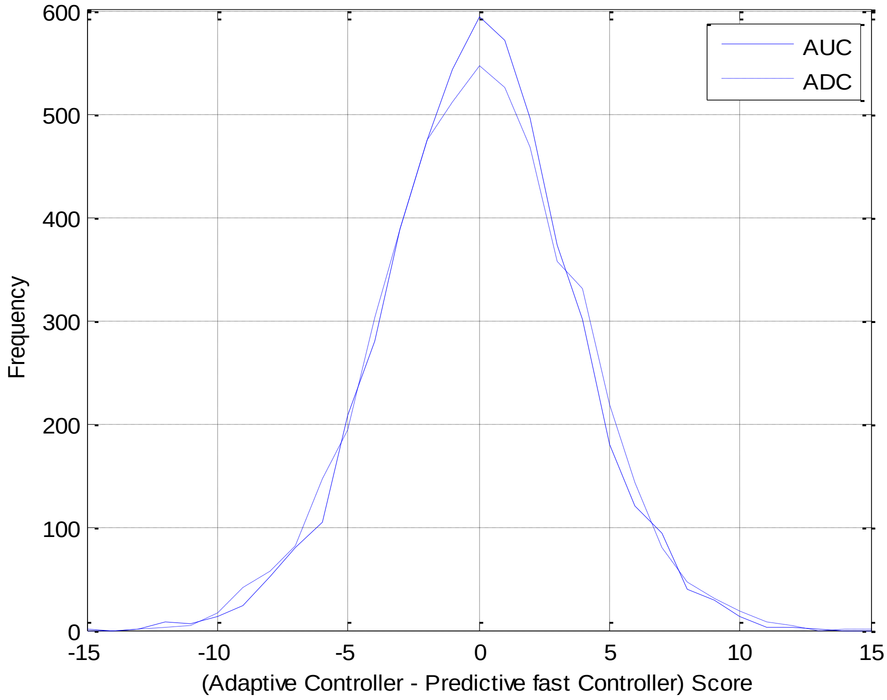
(b)


600
500
400
300
200
100
0
Frequency
-15
-10
-5
0
5
10
15
(Adaptive Controller - Reverse Controller) Score
AUC
ADC

(c)



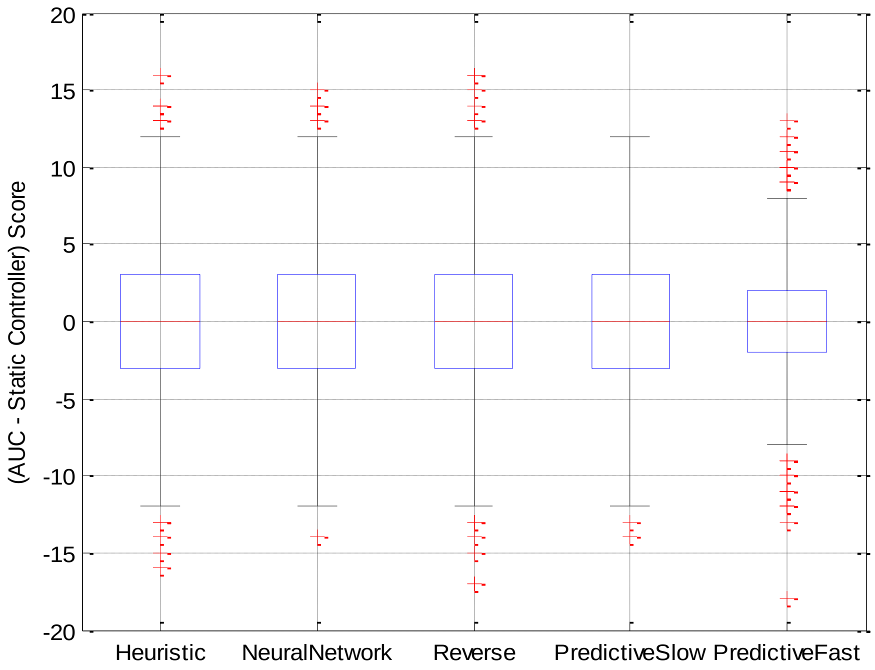
(d)



(e)

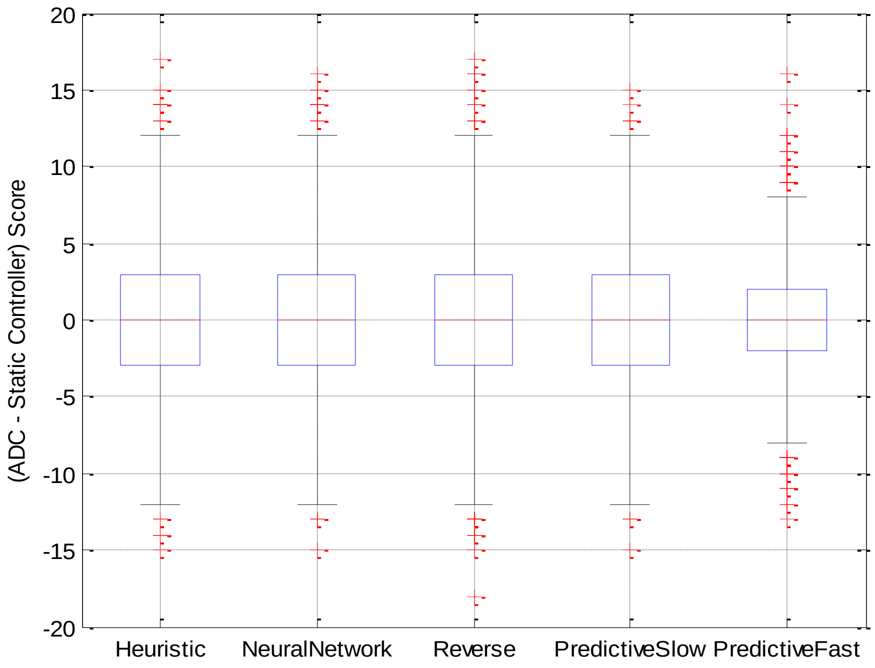
A positive score difference indicates that the adaptive controller won; while a negative score difference indicates that the static controller won.

Figure 5.8 Histogram of the score difference of the adaptive controllers against the (a) HC (b) NNC (c) RC (d) PSC and (e) PFC



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.

Figure 5.9 Boxplot of the results from playing the AUC against the five static controllers



The results are shown in terms of score differences between the ADC and its opponent. A positive score difference indicates that the ADC won; while a negative score difference indicates that the opponent won.

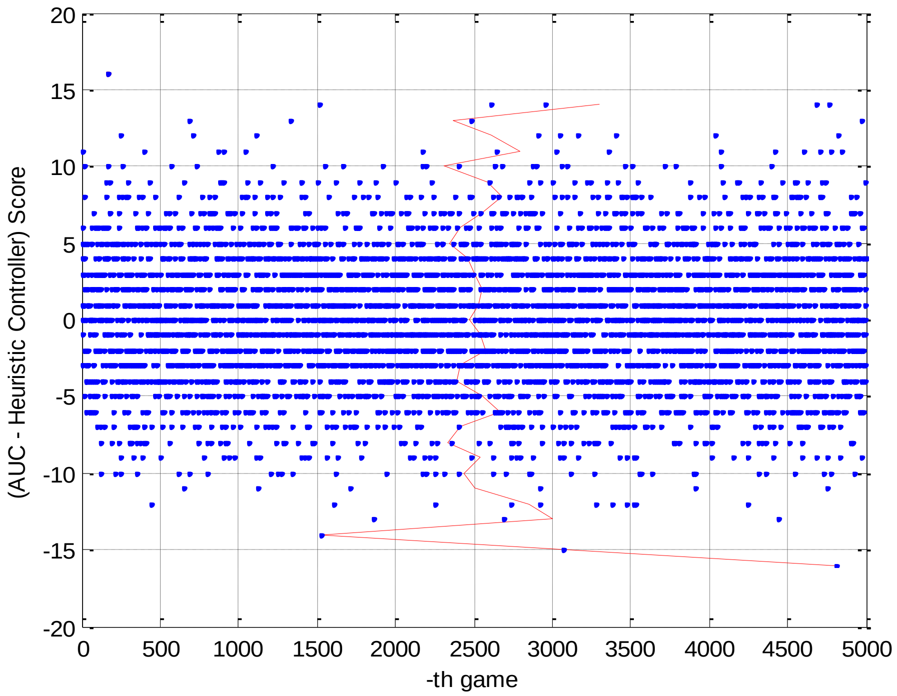
Figure 5.10 Boxplot of the results from playing the ADC against the five static controllers

Table 5.5 Cumulative percentages of games according to score difference

Score diff	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

It was observed in Figure 5.9 and Figure 5.10 that the median values of both AUC and ADC against all their opponents were zero. It was also observed from Figure 5.8 that, by the near symmetry of the histogram, their mean values were very close to zero. These implied that both adaptive controllers were able to match their opponents in terms of mean score difference and winning percentage difference. The upper and lower quartiles of both adaptive controllers were 3 and -3 respectively for HC, NNC, RC and PSC. It was 2 and -2 respectively for the PFC. In addition, it was observed from Table 5.5 that a minimum of 70.22% of the game results between the adaptive and static controllers had a score difference of 4 or less. This indicated that the opponent was entertained in at least 70.22% of the games played.

It was observed from Figure 5.8 that the ADC was likely to have a lower number of drawn games compared to the AUC. Having a lower number of drawn games was a desirable effect as drawn games were deemed to be more frustrating than fun. Hence, the ADC had the advantage of being able to consistently produce a low frequency of drawn games against varied and unknown opponents.

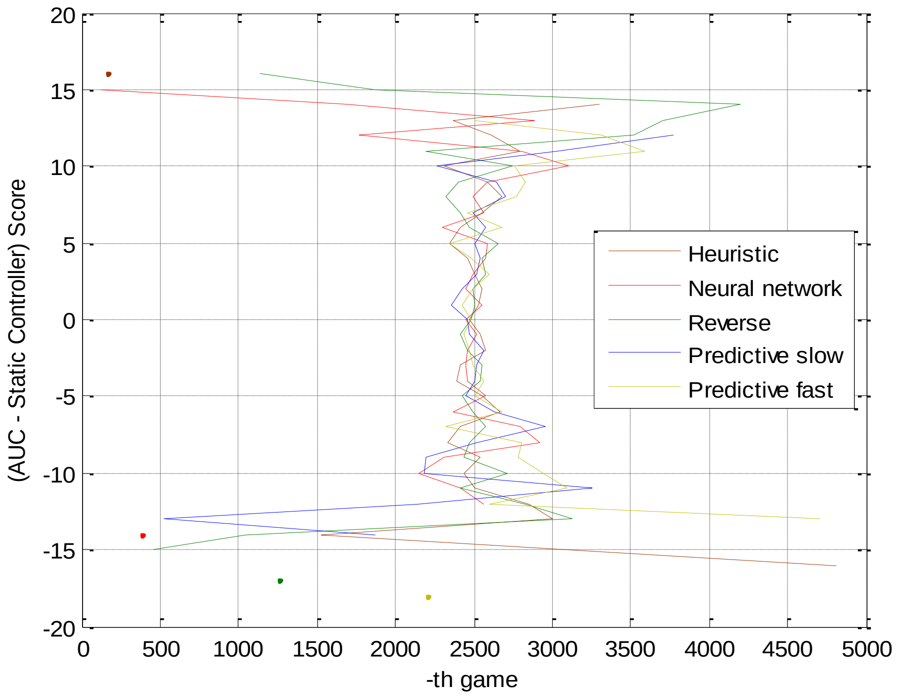


The dots represent an scattered data points of the score difference plotted against the game count. The line represents the mean occurrence of the score difference.

Figure 5.11 A sample diagram of 5000 games between the AUC and HC

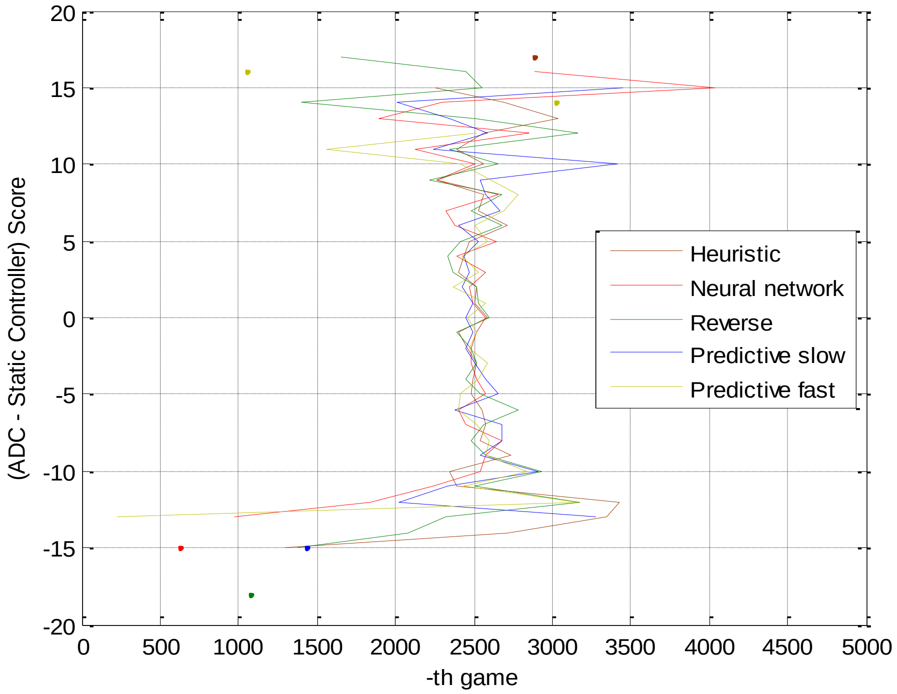
5.4.5.2 Distribution of the occurrence of the score difference

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 2 points as well as lose by 2 points regularly and uniformly over the 5000 sequential games. As an extreme example, the opposite would be to win the first 2500 games by 2 points and lose the last 2500 games by 2 points. In this example, the mean score difference and winning percentage difference is 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 HC is shown in Figure 5.11.



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.

Figure 5.12 Plot of the score difference between the AUC and the static controllers



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.

Figure 5.13 Plot of the score difference between the ADC and the static controllers

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 centre line). This would imply that the particular score difference is evenly distributed. It was observed from Figure 5.11 that the mean occurrence values for score differences of more than 5 tend to diverge away from the centre 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 less will be considered for this analysis.

The results of all the games for the AUC against the five static controllers are plotted together in Figure 5.12. The individual game outcomes are left out to make the diagram more reader friendly. Only the mean value lines are plotted. The results for ADC are shown in Figure 5.13. It was observed that both adaptive controllers had consistent and near zero mean occurrences for score differences in the range of -5 to 5. This indicated that both adaptive controllers were able to evenly distribute varying score differences across a long run of sequential games. This helped to keep the opposing player interested in the game by uniformly winning and losing.

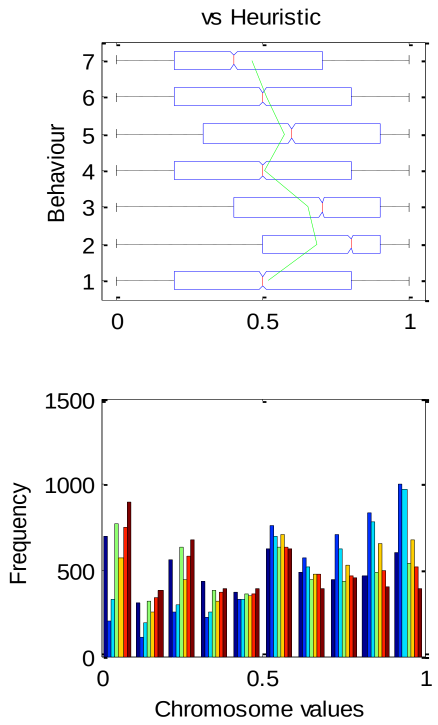
5.4.6 Behaviour activation probability distribution

In this section, the final values of the behaviour 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 behaviours when the adaptive controllers are playing against varied opponents.

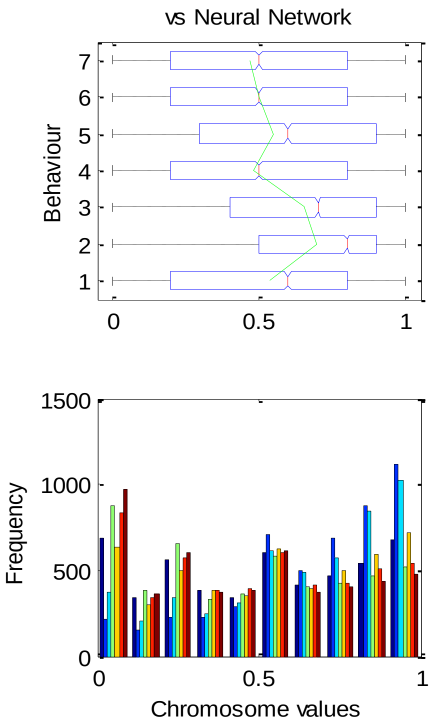
5.4.6.1 Analysis of AUC

The AUC contains one chromosome which encodes 7 real numbers, one for each of the 7 behaviours. Each value represents the probability of activating a specific behaviour. The expected behaviour set encoded by this chromosome represents a ‘winning’ strategy while its complement is assumed to be a losing strategy. The boxplots and histograms of each of the 7 behaviours plotted against the five static opponents are presented in Figure 5.14. The line plotted across the boxplots connects the mean values of each behaviour component. The histogram consists of 10 bins with an interval of 0.1 from 0 to 1. For each bin, the frequency of each behaviour component is represented by a different colour bar.

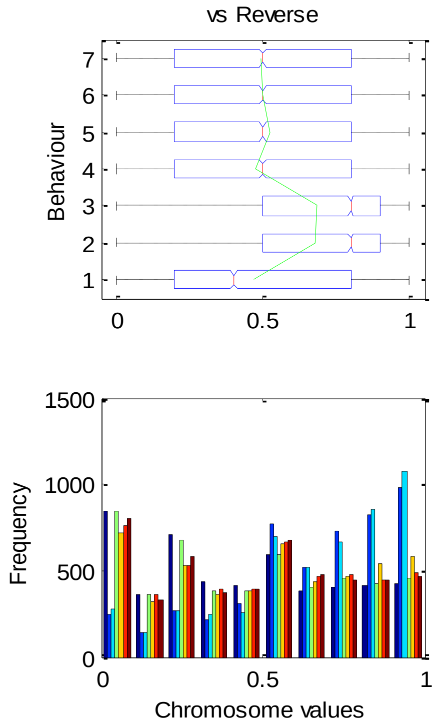
It was observed from Figure 5.14 that there were some general trends in the chromosome values encoded by the AUC at the end of each game. The reversing and direction switching behaviours were selected with high probabilities against all its opponents, indicating that these two behaviours were important behaviours to choose if the AUC wanted to express a winning strategy. The tight angle turning and time wasting behaviours had means and medians of around 0.5 against all opponents. This implied that these two behaviours were not as significant in deciding whether or not a strategy was a winning one.


vs Heuristic
Behaviour
1
2
3
4
5
6
7
0
0.5
1
Frequency
1500
1000
500
0
0
0.5
1
Chromosome values

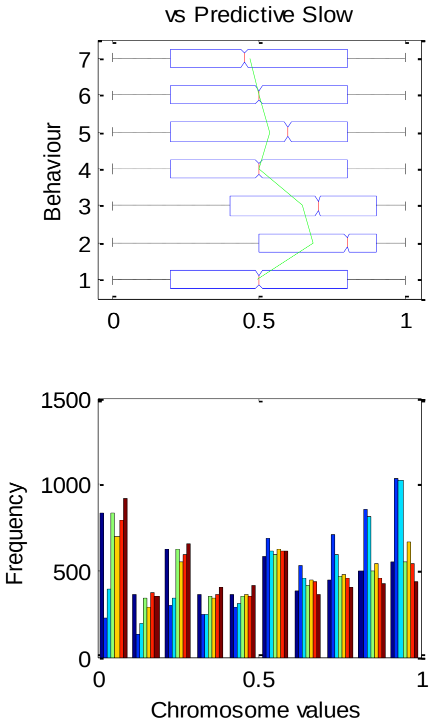
(a)


vs Neural Network
Behaviour
1
2
3
4
5
6
7
0
0.5
1
Frequency
1500
1000
500
0
0
0.5
1
Chromosome values

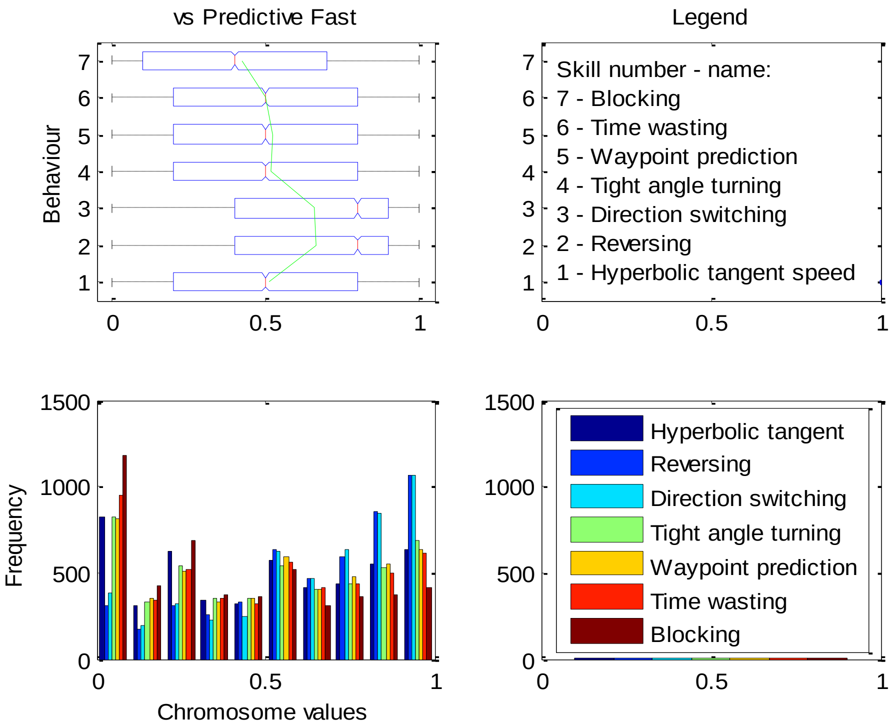
(b)


vs Reverse
Behaviour
1
2
3
4
5
6
7
0
0.5
1
Frequency
1500
1000
500
0
0
0.5
1
Chromosome values

(c)


vs Predictive Slow
Behaviour
1
2
3
4
5
6
7
0
0.5
1
Frequency
1500
1000
500
0
0
0.5
1
Chromosome values

(d)



(e) (f)

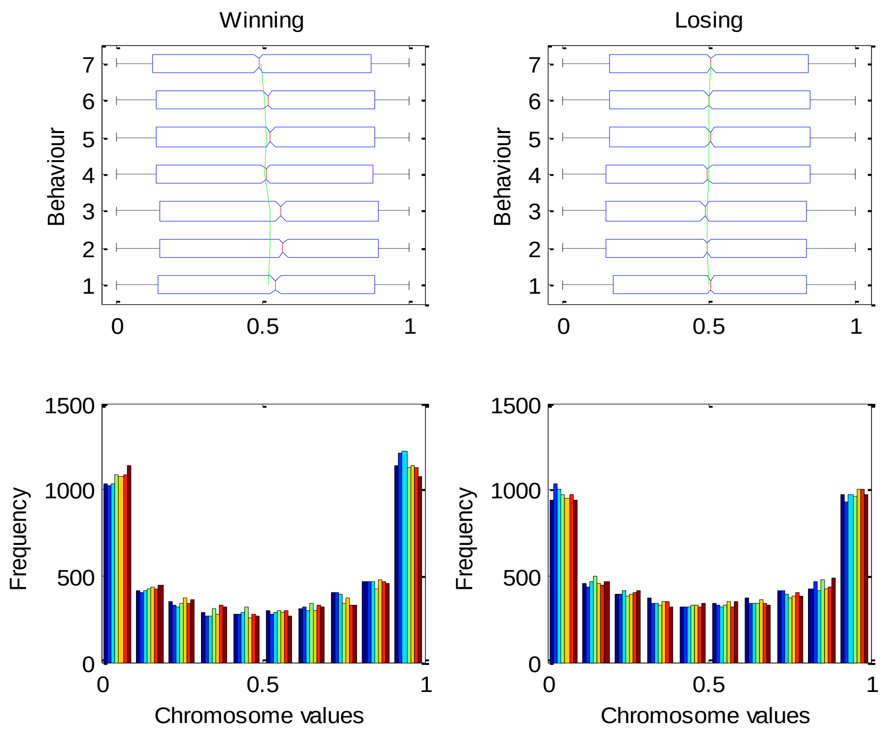
The vertical axis of the boxplot represents the behaviour modules; while the horizontal axis represents the chromosome values in the range [0,1]. The green line connects the mean value of each behaviour modules. The histogram consists of ten bins; the legend for the histogram is shown in (f).

Figure 5.14 Boxplot and histogram of ending chromosome values of the AUC against the (a) HC (b) NNC (c) RC (d) PSC and (e) PFC

The blocking behaviour was selected with the lowest probability amongst the other behaviours. This was because blocking an opponent during a game was a defensive behaviour used to prevent the opponent from getting a point rather than to gain a point for itself. Hence, the AUC generally assigned a lower probability of activation for this behaviour in its chromosome which encoded a winning strategy. This also helped to demonstrate an advantage of the proposed adaptive algorithm in that it was able to select a suitable subset of behaviours automatically via its chromosome. Conflicting behaviours were selected with lower probabilities while complementary behaviours were selected with higher probabilities.

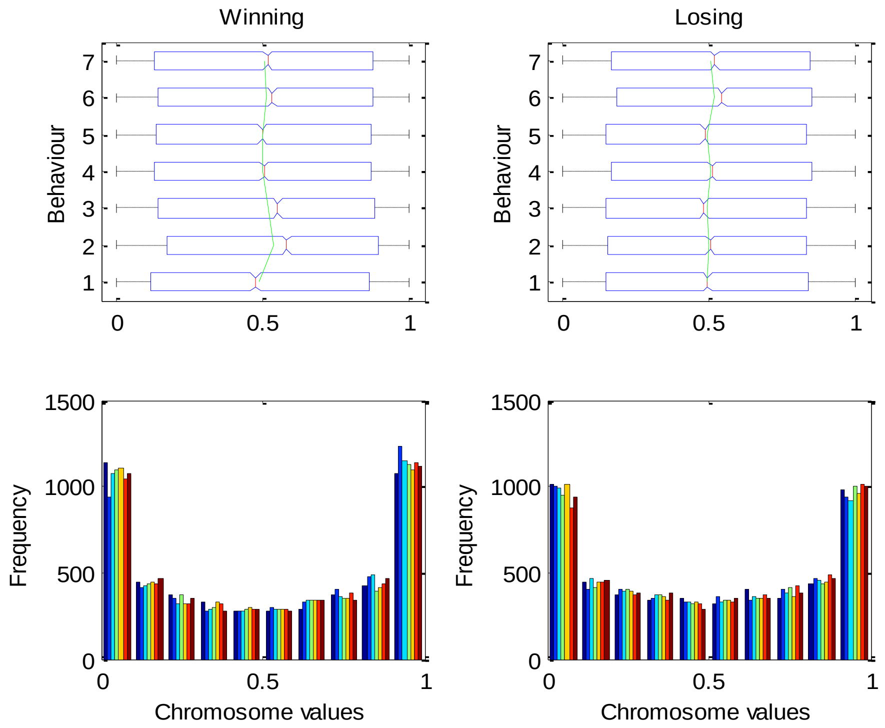
5.4.6.2 Analysis of ADC

The ADC consists of two chromosomes instead of one, each encodes 7 real numbers. One of the chromosomes represents a 'winning' strategy while the other represents a 'losing' strategy. The boxplots and histograms of each of the final value of each of the chromosomes are presented in Figure 5.15 to Figure 5.19.



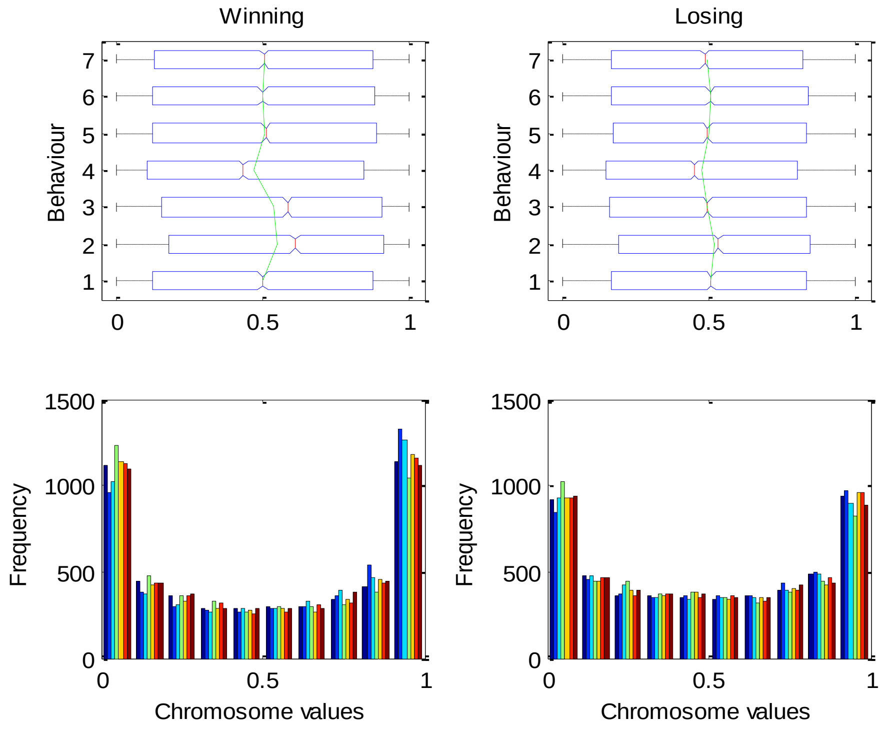
The winning and losing chromosome are shown on the left and right column respectively. The vertical axis of the boxplot represents the behaviour modules; while the horizontal axis represents the chromosome values in the range [0,1]. The histogram consists of ten bins.

Figure 5.15 Boxplot and histogram of ending chromosome values of the ADC against the HC



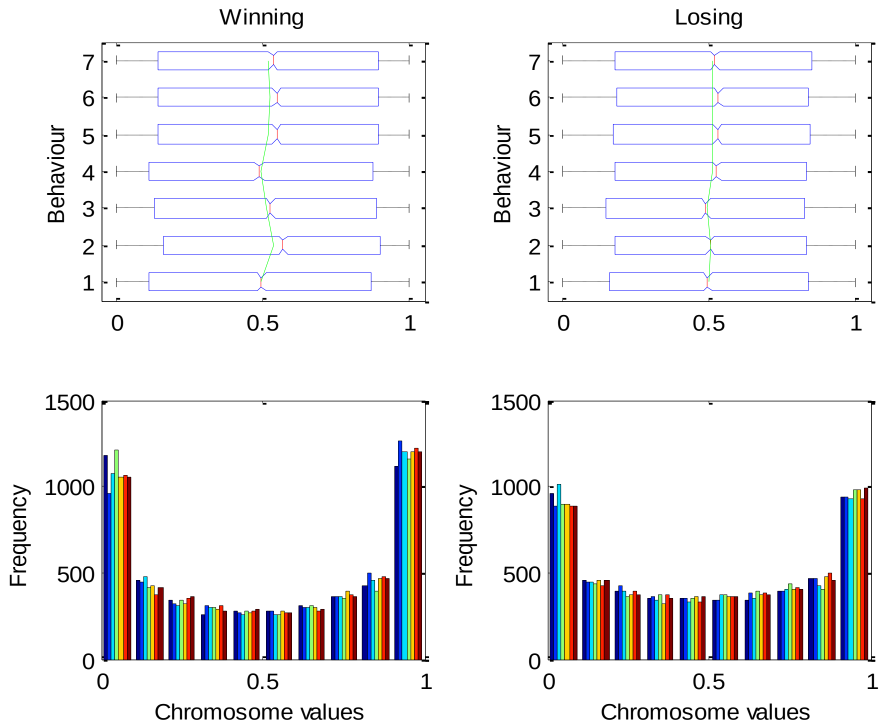
The winning and losing chromosome are shown on the left and right column respectively. The vertical axis of the boxplot represents the behaviour modules; while the horizontal axis represents the chromosome values in the range [0,1]. The histogram consists of ten bins.

Figure 5.16 Boxplot and histogram of ending chromosome values of the ADC against the NNC



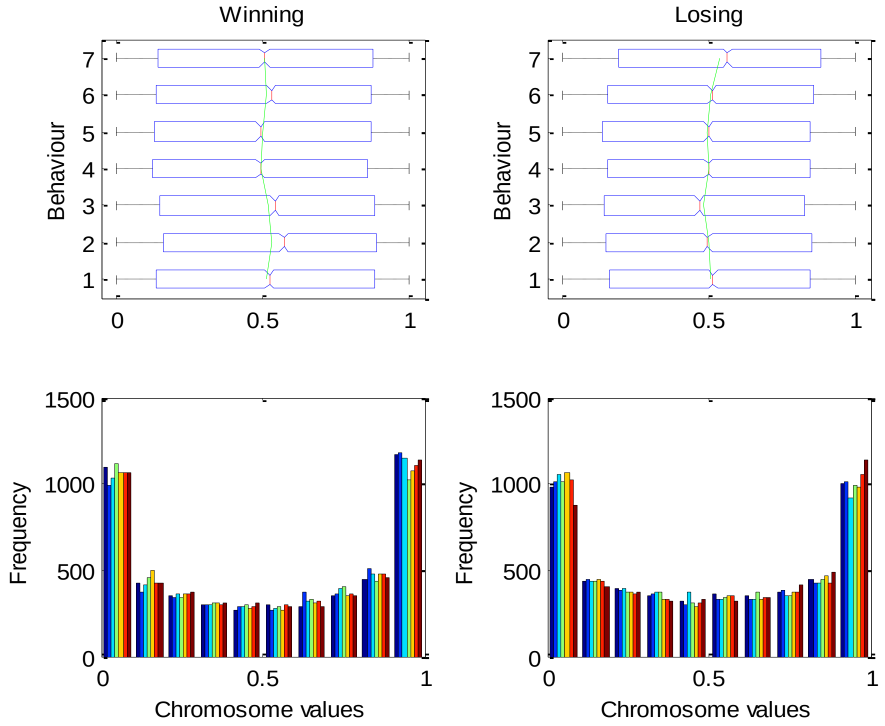
The winning and losing chromosome are shown on the left and right column respectively. The vertical axis of the boxplot represents the behaviour modules; while the horizontal axis represents the chromosome values in the range [0,1]. The histogram consists of ten bins.

Figure 5.17 Boxplot and histogram of ending chromosome values of the ADC against the RC



The winning and losing chromosome are shown on the left and right column respectively. The vertical axis of the boxplot represents the behaviour modules; while the horizontal axis represents the chromosome values in the range [0,1]. The histogram consists of ten bins.

Figure 5.18 Boxplot and histogram of ending chromosome values of the ADC against the PSC



The winning and losing chromosome are shown on the left and right column respectively. The vertical axis of the boxplot represents the behaviour modules; while the horizontal axis represents the chromosome values in the range [0,1]. The histogram consists of ten bins.

Figure 5.19 Boxplot and histogram of ending chromosome values of the ADC against the PFC

It was observed from the histogram in Figure 5.15 to Figure 5.19 that both winning and losing chromosomes was likely to produce high frequencies for the two bin values nearest to 0 and 1. This was likely due to the positive reinforcement nature of the update rules used in the ADC. Behaviours that resulted in winning a waypoint was updated only to the winning chromosome while behaviours that resulted in losing a waypoint was updated only to 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 resembled that of the chromosome from the AUC. This was expected as the single chromosome from the AUC encodes a winning strategy as well. There were no observable trends in the losing chromosome. This might indicate that there were many possible combinations of losing behaviours and that the algorithm learnt a different one each time.

5.5 Summary

Two adaptive algorithms were introduced in this chapter to enhance player satisfaction, namely, AUC and ADC. 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 was examined and both algorithms were able to achieve a score difference of 4 or less 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 was also observed that while the AUC was more computationally efficient, the ADC was able to maintain a lesser number of drawn games which may help to reduce player frustration. In the examination of the ending values of the chromosomes, it was found that the adaptive algorithms select different combinations of behaviours to cope with different opponents although the reversing and direction switching behaviours were observed to be more prominent in winning chromosomes. Both proposed adaptive algorithms were able to automatically learn suitable sets of behaviours 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 opponent driving styles.

Chapter Six

6 Evolving believable behaviour in games using sensor noise and action histograms

A believable game AI can help players to immerse in the game world and maintain the suspension of disbelief, thereby making the game more enjoyable and satisfying. This chapter explores the use of two main ideas to acquire believable behaviours. First, sensor noise is introduced to simulate errors in human judgment and its associated parameters are evolved together with the game controller. Second, indirect modeling of human behavioural tendencies is achieved by using output action histograms as optimization objectives. Two types of histograms will be explored, the action histogram and the action sequence histogram. The proposed approach differs from conventional approaches by focusing on imitating actions within a small window size instead of imitating the entire action sequence. The resulting controllers with evolved sensor noise are able to achieve both objectives of performance and believability in training, and demonstrate good generalization capability on 4 other previously unseen test tracks. In a study involving 58 respondents, the same controllers are also evaluated as more believable compared to one evolved for performance alone.

6.1 Introduction

Computational intelligence design methodology has seen an increase in its application to games in the recent decade. Techniques such as neural networks, fuzzy logic and evolutionary computation have been applied to design and develop game artificial intelligence (AI) for game characters. Often, these game characters are adversarial in nature and seek to outplay and defeat its opponent in a game. As such, research involving computational intelligence and games is traditionally concerned with playing a game as well as possible. Most often the objectives are to get the highest score, fastest time or to defeat the opponent. A few examples are given as follows. Chellapilla & Fogel evolved neural networks to play checkers and was able to defeat two expert level players on an internet game room [32]. Stanley et al used the real time NeuroEvolution of Augmenting Topologies method to evolve a team of robots to defeat opposing robot teams [152]. Spronck and Spronck et al used the dynamic scripting algorithm to adaptively optimize the game performance of opponents in a role playing game [147] [148]. Togelius & Lucas evolved controllers that were able to exhibit good racing behaviour in a car racing simulator [171].

Indeed, a competent game AI is an important factor in enhancing the gaming experience of the player [49], but it is not the only factor. Game designers also want their players to immerse in the game world and suspend disbelief [158], thereby making the game more enjoyable and satisfying. As such, the field of computational intelligence in games has seen the emergence of more player centric works in the recent years that focused on improving a human player’s experience [164]. Togelius et al evolved personalized racing tracks that catered to the driving styles of different human players [172] [176]. Tan et al implemented online driving adaptation to match the proficiency level of its opponents [161]. Spronck et al adapted the dynamic scripting algorithm to play even games against its opponents [145]. van Lankveld et al introduced incongruity as a potential measure for entertainment [188] [189]. Yannakakis used evolutionary machine learning to exploit cooperative behaviours that can increase a player's interest while playing [194]. Yannakakis et al also implemented an adaptive Bug-Smasher game that improved the satisfaction of children who played the game [198] [199]. Thue et al used an interactive storytelling system that models a player automatically to dynamically select content to create an interactive story [168] [169]. Pedersen et al optimized the level design of platform games for improving player experience [109] [110] [111] using Super Mario Bros [181]. Choi et al and Langley et al outlined an approach to constructing believable game players games using a cognitive architecture [34] [82]. Sweetser & Wiles developed game agents that were able to respond believably to the environment [159]. Miles & Tashakkori evolved a more believable game agent using genetic algorithms compared to using traditional finite state machines [95].

An important area of player centric research deals with the creation of believable game agents. Game agents that are believable can help to maintain the suspension of disbelief and build a more immersive game world that can improve the player's satisfaction in a game [19]. For example, Rizzo et al implemented a personality model for agents to perform personality driven behaviours [129]. Computational intelligence techniques can also be used to acquire such believable game behaviours. Bryant & Miikkulainen evolved neural networks to induce game agents with human similarity in a turn based strategy game [25]. Thurau et al applied imitation learning for a first person shooting game to learn strategic, tactical and reactive behaviours [170]. van Hoorn et al evolved humanlike driving behaviours in the TORCS game by imitating steering and acceleration data from human players [187]. Yannakakis provided a review on several approaches used to model player satisfaction [197].

Imitation learning has indeed been demonstrated as a powerful tool for learning many types of complex behaviours [9] including game behaviours. Cardamone et al developed an approach to imitate high level actions in TORCS to improve driving performance [29]. Muñoz et al used artificial neural networks to train controllers that imitate humans and other AI [98]. Priesterjahn & Eberling used imitation and social learning to quickly generate competitive game agents [117]. Aler et al used imitation learning to train game agents for Robosoccer [3]. So far, the works in literature shared a common approach in imitation learning. Data collected from human players is used as training data in the form of state-action pairs. That is, the learning agent is trained to imitate the decision (i.e. in-game output action) of the human player for a given situation (i.e. game state or sensor readings). The learnt agent thus exhibits humanlike-ness because it reacts in the same way as the human player for a given situation. In this chapter, a different approach will be introduced.

Using the same idea of imitation learning, instead of imitating state-action pairs, the imitation of human behavioural tendencies will be considered as a means to induce humanlike-ness in generic agents. As a metaphoric example, instead of learning how to reply questions, the game agent learns how to blink his eyes in a believable manner while replying. Believability is achieved through the learning of such low level human tendencies or idiosyncrasies. This chapter focuses on the evolution of believable behaviours in games using a combination of sensor noise and action histograms. Two main ideas will be explored in this chapter. First, some noise is introduced to the sensors of the game AI in order to imitate errors in human judgment. The parameters associated with the noise are evolved simultaneously with the game AI to allow the evolution process to discover what values are suitable to induce believable behaviours. Second, the output action histograms and output action sequence histograms are introduced as a means to capture low level behavioural tendencies of the game agent. This is motivated by the observation that previously evolved car controllers [166], which drove in unconvincing manners, have very different histograms when compared to the histograms collected from human driving. Hence, the car controllers will be trained to indirectly model human driving in terms of histogram instead of directly modeling state-action pairs in the game. Concurrently learning of both performance and believability requires the use of multi-objective evolution. Multi-objective evolution has been successfully applied to games to introduce other desirable objectives in addition to basic performance. van Hoorn et al evolved performance while imitating human drivers [187], Gomez et al evolved performance with behavioural complexity [56], and Agapitos et al used multi-objective optimization to evolve car drivers with different driving style [2]. The multi-objective evolution framework will be used in this work to balance the two incomparable and partially conflicting objectives of performance and believability via action histograms. Although the proposed training methodology is demonstrated in a car racing simulator game, the framework can be easily extended to games of similar control schemes such as platform games (i.e. Super Mario, Rayman, etc) and arcade shooters (i.e. Asteroids, Space Invaders, etc). This is the first work to explore the use of action histograms and evolvable sensor noise as a means to develop believable behaviours.

6.1.1 Modifications to simulator

The discrete control scheme that is used in the car racing simulator provided a suitable test bed for capturing low level behaviours using the proposed action histograms. Nevertheless, the ideas introduced in this chapter can be extended to similar discrete control games such as platform games and arcade shooters.

For the purpose of this experiment, a few changes are made to the car racing simulator. The car racing simulator is modified to be played by only 1 player and only the current waypoint will be visible to the player. This is so as to focus on basic driving behaviours instead of predictive and planning abilities. The same simulator will be played by human testers in this experiment.

At any time, only one waypoint is visible on the competition field, the current waypoint. The player must drive through this waypoint in order to score a point. Whenever the current waypoint is passed, 1 point will be added to the total score and a new waypoint will be generated. The position of each waypoint is randomly generated anywhere within the boundaries of the game area. The random number generator can also be seeded with a fixed integer in order to generate a fixed sequence of waypoints which can be used to define a repeatable race track.

6.2 Controller design

The design of the car racing controller will be described in this section. The controller consists of 2 sub-modules, one for controlling the accelerating and reversing behaviour of the car while the other controls the steering behaviour of the car. Both sub-modules use the same hyperbolic tangent function as a basis function.

6.2.1 Hyperbolic tangent driving

The hyperbolic tangent driving function is the first of 2 sub-modules that controls the behaviour of the car. As the name suggests, this sub-module controls the speed of the car by issuing an accelerate command, a brake command, or a neutral command. It decides which command is issued at every time step by comparing the instantaneous speed of the car to the desired speed of the car which is defined by the equation for the hyperbolic tangent driving function in (6.1).

$$v(r) = a \times \tanh\left(b \times \left(r + rZ_1\right) + c\right) + d \qquad (6.1)$$

where r is the Euclidean distance to the current waypoint, a, b, c and d are real value parameters characterizing the hyperbolic tangent function, v is the desired scalar speed at a given Euclidean distance r, and Z_1 is a noise variable. The 4 parameters a, b, c and d will be optimized by evolution. The hyperbolic tangent function is chosen because of its general shape. The tapering of its outputs at high values of r is analogous to the notion that the car should cruise at a constant speed at far distances from its destination (i.e. the cruising speed should not increase indefinitely with distance). Additionally, the steep gradient around the origin is analogous to deceleration when it is near the destination. The values a, b, c and d serve to shape the hyperbolic tangent function to one most desirable for this car racing simulation. There are no constraints that the function; e.g. it does not need to pass through the origin or that it should be positive or negative.

At each time step, the controller will calculate the desired speed of the car using the hyperbolic tangent driving function, and compare it to its instantaneous speed. The command to accelerate, reverse or remain neutral will then be decided based on the rules in (6.2).

$$O_d\left(v_i\right)=\begin{cases} accelerate & if \quad v_i+v_iZ_2>\left(1+n_d\right)v, \\ brake & if \quad v_i+v_iZ_2<\left(1-n_d\right)v, \\ neutral & otherwise. \end{cases} \tag{6.2}$$

where O$_d$ is the output of the driving module, v$_i$ is the instantaneous speed of the car, v is the desired speed, n$_d$ is a real number, and Z$_2$ is a noise variable. The purpose of n$_d$ is to provide a margin of allowable difference between O$_d$ and v$_i$ within which the instantaneous speed is considered to be desirable and a neutral command is issued. The parameter n$_d$ will be optimized by evolution and it is also unconstrained.

6.2.2 Hyperbolic tangent steering

The hyperbolic tangent steering function is the second of 2 sub-modules that controls the behaviour of the car. This sub-module determines the heading of the car by issuing a steer left, steer right or neutral command. It decides which command is issued at every time step by comparing the instantaneous angular speed of the car to the desired angular speed of the car which is defined by the equation for the hyperbolic tangent steering function in (6.3).

$$\omega(\theta) = e \times \tanh\left(f \times (\theta + rZ_3) + g \right) + h \tag{6.3}$$

where θ is the angular distance to the current waypoint, e, f, g and h are real value parameters characterizing the hyperbolic tangent function, ω is the desired angular speed at a given angular distance θ, and Z_3 is a noise variable The 4 parameters e, f, g and h will be optimized by evolution. The design of the steering function is similar to that of the accelerating-braking function except that the input is replaced by an angular distance and the output gives the desired angular speed. In a similar vein, the hyperbolic tangent function is chosen because it is natural for the car to be steering strongly when it is not aligned with its destination (i.e. large angular distance to the current waypoint), and also to reduce its steering action as it aligns with its destination. The function should pass through the origin as the desired angular speed should be zero (i.e. straight) when the angular distance is zero (i.e. exactly aligned to its destination). However, no constraints are placed on the parameters e, f, g and h as the evolution process is expected to find such a solution as an optimum solution.

At each time step, the controller will calculate the desired angular speed of the car using the hyperbolic tangent steering function and compare it to its instantaneous angular speed. The command to turn left, right or remain neutral will then be decided based on the rules in (6.4).

$$O_s(\omega_i) = \begin{cases} \mathit{left} & \mathit{if} \quad \omega_i + v_i Z_4 > (1 + n_s)\,\omega, \\ \mathit{right} & \mathit{if} \quad \omega_i + v_i Z_4 < (1 - n_s)\,\omega, \\ \mathit{neutral} & \mathit{otherwise.} \end{cases} \tag{6.4}$$

where O_s is the output of the steering module, ω_i is the instantaneous angular speed of the car, ω is the desired angular speed, n_s is a real number, and Z_4 is a noise variable. Similar to that of the driving module, the purpose of n_s is to provide a margin of allowable difference between O_s and ω_i within which the instantaneous angular speed is considered to be desirable and a neutral command is issued. Similarly, the parameter n_s will be optimized by evolution and it is also unconstrained.

6.2.3 Introducing sensor noise

The car racing simulator model is a fully deterministic model. That is, a controller will output the same sequence of driving and steering commands and trace exactly the same trajectory as long as the positions of the sequence of waypoint remain the same. This is both uninteresting and unrealistic. In a realistic simulation, for example, humans tend to be able to judge distances better when the subject is in close proximity. When the subject is far away, the error in judgment also becomes larger.

As such, noise is introduced to the sensors of the vehicles in the car racing simulator model to make the simulation more stochastic and realistic. The sensor noise being introduced to the system takes the form of additive Gaussian noise, Z, with mean, μ, and standard deviation, σ, which follows the normal distribution given in (6.5).

$$Z \sim N\left(\mu,\sigma^2\right) \tag{6.5}$$

The mean and standard deviations of the Gaussian noise is not specified by design. Instead, these values are evolved together with the controller by evolution strategies. The idea is to allow the evolution process to discover what combinations of sensor noises will result in humanlike behaviours.

The choice of using Gaussian noise in the noise model is inspired by the study of measurement uncertainty. Measurement errors made by humans are divided into two components, systematic error and random error. A basic type of systematic error is caused by the incorrect calibration of the measuring instrument. This error is constant and always present in separate measurements. The mean of the Gaussian noise introduced to the controller is analogous to systematic errors. On the other hand, random errors are inconsistent in repeated measures and tend to be scattered about the true value. Random errors can be caused by imprecise instruments or subjective interpretation of the instrument reading by the user, and this is analogous to the standard deviation of the Gaussian noise introduced to the controller. Therefore, these reasons make Gaussian noise a suitable choice to imitate errors in human judgment.

Sensor noise is introduced to the four sensors that are used in the driving and steering sub-modules. They are the Euclidean distance to the current waypoint, the instantaneous speed of the car, the angular distance to the current waypoint, and the instantaneous angular speed of the car. The addition of these sensor noises modifies the behaviour of the hyperbolic tangent driving and hyperbolic tangent steering components.

$$Z_j \sim N\left(\mu_j, \sigma_j^2\right) \quad for\ j = \{1,2,3,4\} \tag{6.6}$$

The additive Gaussian noise variables Z_1, Z_2, Z_3 and Z_4 shown in equation (6.6) are added to the sensor values shown in equations (6.1), (6.2), (6.3) and (6.4) respectively. These noise variables are also modified by a corresponding coefficient depending on the state of the car in the game. Noise variables Z_1 and Z_3, which affects the sensing of the distance and angular distance respectively, are modified by the real distance between the car and the next waypoint. That is, more noise is added when the car is far away from the next waypoint; and less noise is added when the car is near its next waypoint. In other words, the amount of noise decays with decreasing distance. It should be noted that modern hardware sensors, such as those used in robotics, do not behave in this way. The noise decay is introduced in the noise model to imitate human judgment in a gaming environment. This is in line with the result that the error in judging distances in a virtual environment increases with distance [20]. In the model, the relationship is assumed to be linear for simplicity. This concept is also extended to speeds and judging speeds. Similarly, noise variables Z_2 and Z_4, which affects the sensing of speed and angular speed, are modified by the real instantaneous speed of the car. That is, more noise is added when the car is moving quickly and less noise is added when the car is moving slowly. For simplicity, the relationship is also assumed to be linear.

In this chapter, the use of evolvable sensor noise will be explored as a means to simulate human judgment errors in order to improve the believability of the evolved controllers. The parameters that are used to define the sensor noise are evolved alongside the parameters that define the driving and steering components of the controller so as to allow the evolution process to discover the optimal amount of noise required to improve believability.

6.3 Action histograms

In earlier experiments with developing a controller for the car racing simulator, the controllers were evolved using a single objective approach with the fitness function defined by the number of waypoints passed within a given number of time steps. The resulting controllers were able to competently drive around the race area collecting waypoints and also react to situations that were unseen during the training phase. It could be said that the evolved controllers were robust and were able to generalize well. However, the visually observed behaviour of these controllers was unnatural and unrealistic.

In order to investigate the differences in observed behaviours, the output actions of the Evolved Heuristic controller (EH) [166], Evolved Neural Network (ENN) controller [166] and Human (Hu) are quantitatively and qualitatively analyzed in this section. During each time step in the simulation, each controller is required to output one of the nine possible actions presented in Table 6.1. The set of output actions used by each controller on the same given track is collected and presented in the form of histograms. In effect, this measures the frequency with which a keystroke is being used when driving around a given track.

This section will introduce and discuss two types of histograms. They are the action histogram and the action sequence histogram. The experimental procedure for data collection and the types of tracks used will also be discussed and analyzed. Finally, the motivations for using histograms of small window sizes will be presented.

Table 6.1 List of all possible output actions at each time step in the car racing simulator

Action	Description
1	Reverse Left
2	Reverse
3	Reverse Right
4	Left
5	Neutral
6	Right
7	Forward Left
8	Forward
9	Forward Right

6.3.1 Action histogram (Histo1)

The action histogram is the histogram of the set of n output actions used by a controller during a simulation of n time steps. There are 9 possible output actions during each time step. Hence, the action histogram contains 9 bins, as given in equation (6.7).

$$\sum_{i=1}^{9} m_i = n \tag{6.7}$$

where n is the total number of observations, in this case n = 1000, and m_i is the number of observations that fall into bin i. The action histogram can be thought of as a histogram of the output actions with window size one.

6.3.2 Action sequence histogram (Histo2)

The action sequence histogram is the histogram of the set of n-1 transitions of sequential output actions used by a controller during a simulation of n time steps. That is, the action sequence histogram can be thought of as a histogram of output actions with window size two. For a set of 9 possible output actions, there are 81 possible transitions of output actions. Hence, the action sequence histogram contains 81 bins, as given in equation (6.8).

$$\sum_{i=1}^{81} m_i = n - 1 \tag{6.8}$$

where n is the total number of observations, in this case n = 1000, and m_i is the number of observations that fall into bin i. Action sequence a-b falls into bin (9a-9+b). The action histogram can be thought of as a histogram of the output actions with window size two.

6.3.3 Data collection

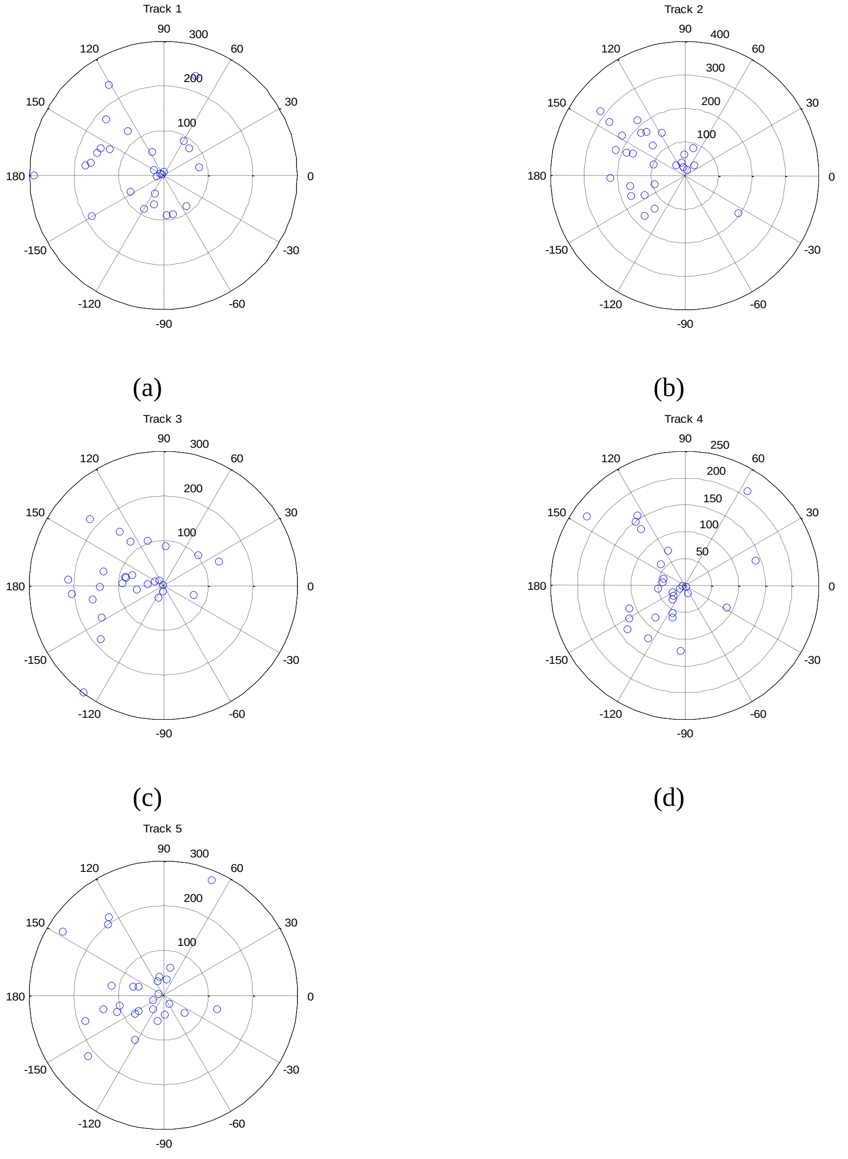
The first task of this experiment was to collect human driving data in order to build the training data to be used in the evolution process. The human player was asked to play the solo version of the car racing simulator game several times to be familiarized with its control mechanisms and game physics. The initial trial runs were conducted on randomly generated tracks and were not recorded.

Next, the human player was asked to drive on a predefined track. At each time step in the simulation, the state of the game was recorded together with the output action from the human player. Each simulation lasted 1000 time steps. The experiment was then repeat 4 more times for a total of 5 sets of data on the same race track. It was necessary to restrict the data collection to 5 trials per track because the human player was able to learn from experience and memorize the position of the next waypoint on a track after a few trials. The entire experiment was then repeated for 4 other predefined tracks for a total of 5 tracks.

The sets of human driving data for each track were then converted to action histograms Histo1 and action sequence histogram Histo2. For each track, the averages were obtained from the 5 sets of collected data.

Table 6.2 Number of waypoints passed by human collected over 5 trials

Human	Score	
	Mean	Std
Track 1	17.8	0.45
Track 2	15	0
Track 3	15.8	0.45
Track 4	16	0
Track 5	17.8	0.84



(a)

(b)

(c)

(d)

(e)

Each point on the polar diagram represents the distance and heading of the current waypoint with respect to the last waypoint.

Figure 6.1 Polar diagram of the waypoints of (a) track 1 (b) track 2 (c) track 3 (d) track 4 and (e) track 5

Table 6.3 Action histograms and action sequence histograms by human collected over 5 trials

Track 1								
Histo1								
215.6	215.6	73.4						
82.8	359.8	52.8						
0	0	0						
Histo2								
188.6	15.4	0	11.4	161.4	5.8	0.2	7.6	56.8
5.2	6	0	1	35	0.6	0	2.8	6
0	0	0	0	0	0	0	0	0
13.6	0.2	0	1.8	30.2	1.4	0	0.8	9.4
57	11.8	0	19.6	292.4	14.4	0	10.8	31.8
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
Track 2								
Histo1								
222	255.6	37						
58.6	399	27.8						
0	0	0						
Histo2								
198.6	16.8	0	9.4	199.4	3.4	0	3.6	31.2
0.8	5.8	0	0.4	42.6	0.2	0	1.8	0.2
0	0	0	0	0	0	0	0	0
13	0.4	0	1	35.2	0.2	0	0.2	2.2
36	9.2	0	21.4	330.2	10.8	0	8.4	16.6
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
Track 3								
Histo1								
140.8	204.4	168.4						
29.8	396.2	60.4						
0	0	0						
Histo2								
128.6	8.2	0	4.6	159.8	3.6	0	10.2	150.2
0.2	3.4	0	0	35.6	0.6	0	4	4
0	0	0	0	0	0	0	0	0
6.6	0.2	0	1	26	0.8	0	0	13.8
17.4	5.6	0	12.2	339.4	16.4	0	7.2	39.4
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
Track 4								
Histo1								
107.2	197	185.2						
27.6	448.6	34.4						
0	0	0						
Histo2								
96.8	8.6	0	2.8	159.4	3	0	9.6	173
0.2	1.6	0	0.4	31	0.4	0	1.2	0.8
0	0	0	0	0	0	0	0	0
5.6	0.4	0	2	18.8	0.2	0	0.2	9
14.6	6.8	0	12.4	402	13	0	5	20.2
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
Track 5								
Histo1								
109.2	240.2	157						
65	376	52.6						
0	0	0						
Histo2								
93.8	8.8	0	2.8	195.8	5.8	0	13.2	141.2
4.8	1.6	0	0.4	35.2	0.2	0	1.6	1
0	0	0	0	0	0	0	0	0
11.4	0.4	0	1.2	21.4	0.2	0	0.6	9.8
44.2	8.8	0	15.6	322.4	14.6	0	5.4	36.8
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

The 5 tracks used in this experiment were randomly generated in a boundary free area. Hence, many portions of each track were overlapping and difficult to visualize even if the routes were plotted. As such, the characteristics of each track are summarized in the form of a polar diagram in Figure 6.1. Each waypoint on a track is characterized by its bearing and distance from the last waypoint. A waypoint that is very near and yet at a large bearing from the previous waypoint will require the car to make a sharp turn to reach it while a waypoint that is very far and have a small bearing offers a chance for the car to accelerate to higher speeds. Therefore, the difficulty of a given track can be characterized by the distribution of its waypoints on the polar diagram. This approach is not as effective as seeing the real layout of a track, but it works as a good compromise given the overlapping nature of the driving routes.

The waypoint scores are presented in Table 6.2 while the histograms Histo1 and Histo2 are presented in table form in Table 6.3. It was observed that human driving used only the actions 4 (Left), 5 (Neutral), 6 (Right), 7 (Forward Left), 8 (Forward) and 9 (Forward Right) for all the 5 tracks. The reversing actions, 1 (Reverse Left), 2 (Reverse) and 3 (Reverse Right) were not used. Human driving data also showed a high percentage of the time spent doing nothing (i.e. action 5) on all tracks. In Histo2, the dominant action sequence was also doing nothing (i.e. 5-5). This was to allow the car to slow down due to friction and also to observe feedback of the effects of its actions during previous time steps before making the next action. The next dominating action sequence was forward acceleration (i.e. 8-8) which was used to drive forward in a straight line.

Only the data from track 1 will be used for training purposes. The remaining tracks will be reserved for testing the generalization capability of the evolved controllers.

6.3.4 Case study

This section presents an analysis, using histograms, of the differences in behaviour between the Human (Hu), Evolved Heuristic (EH) and Evolved Neural Network (ENN) controller. Figure 6.4, Figure 6.5 and Figure 6.6 show the normalized histogram of the output actions of the controllers EH, ENN and Hu respectively. In each figure, high colour intensity (white) indicates a low frequency of usage of the output action while low colour intensity (black) indicates a high frequency of usage.

	7 Forward-Left	8 Forward	9 Forward-Right
Driving	4 Left	5 Neutral	6 Right
	1 Reverse-Left	2 Reverse	3 Reverse-Right

Steering

Figure 6.2 Graphical representation of the action histogram to mimic the layout of arrow keys on the keyboard

Figure 6.4 (a), Figure 6.5 (a) and Figure 6.6 (a) show the simple histogram of the actions taken by the controllers during a game. Figure 6.2 shows how the positions of the squares in the action histograms are interpreted. Figure 6.4 (b), Figure 6.5 (b) and Figure 6.6 (b) show the histogram of the change in sequential actions during a game. Figure 6.3 shows how the positions of the squares in the action sequence histograms are interpreted. For example, if the output action of a controller in the previous time step is 8 (forward) and the output action in this time step is 9 (forward right), then the frequency of the change in action from 8 to 9 is incremented by one. In a sequence histogram, this is represented in Figure 6.3 as the square in the first row from the top, sixth column from the left and labeled “8-9”. In a game of 1000 time steps, there are 999 changes in sequential actions.

Driving

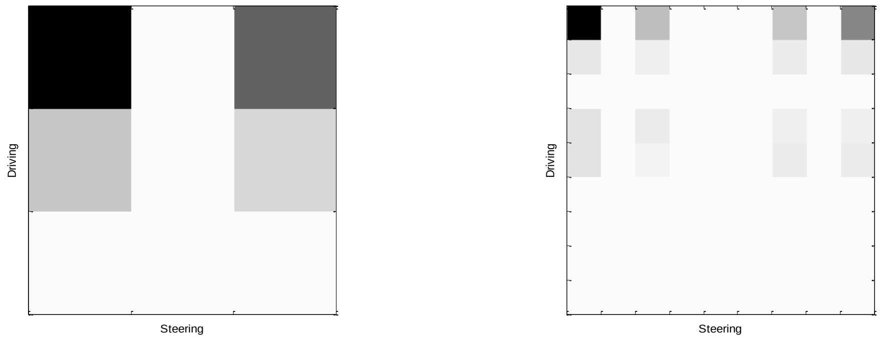
7-7	7-8	7-9	8-7	8-8	8-9	9-7	9-8	9-9
7-4	7-5	7-6	8-4	8-5	8-6	9-4	9-5	9-6
7-1	7-2	7-3	8-1	8-2	8-3	9-1	9-2	9-3
4-7	4-8	4-9	5-7	5-8	5-9	6-7	6-8	6-9
4-4	4-5	4-6	5-4	5-5	5-6	6-4	6-5	6-6
4-1	4-2	4-3	5-1	5-2	5-3	6-1	6-2	6-3
1-7	1-8	1-9	2-7	2-8	2-9	3-7	3-8	3-9
1-4	1-5	1-6	2-4	2-5	2-6	3-4	3-5	3-6
1-1	1-2	1-3	2-1	2-2	2-3	3-1	3-2	3-3

Steering

For example, the sequence 8-8 (Forward-Forward) is obtained by pressing the Up arrow key twice, hence the ‘8’ position of the large grid (single solid line) followed by the ‘8’ position of the small grid (single dotted line).

Figure 6.3 Graphical representation of the action sequence histogram based on the layout in Figure 6.2

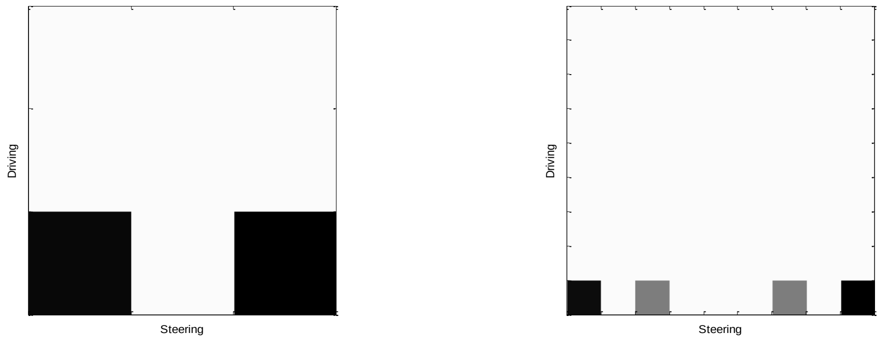
The comparative action histograms and action sequence histograms for tracks 1, 2, 3, 4 and 5 are plotted in Figure 6.7, Figure 6.8, Figure 6.9, Figure 6.10 and Figure 6.11 respectively. First, the histogram of actions of the controllers will be examined for unnatural behaviours. It was observed from Figure 6.4 (a) that EH used only the actions 4 (Left), 6 (Right), 7 (Forward Left) and 9 (Forward Right), and predominantly actions 7 and 9. This meant that in terms of steering, EH was constantly steering either left or right but never neutral. This was counterintuitive as the controllers were expected to naturally drive straight once it had aligned itself to the waypoint.



(a) (b)

Values are normalized. High colour intensity (white) indicates a low frequency of usage while low colour intensity (black) indicates a high frequency of usage.

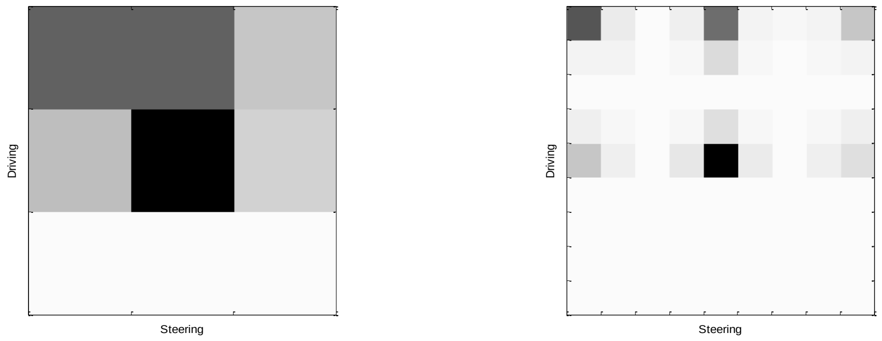
Figure 6.4 Histogram of the (a) output actions and (b) output action sequences of the EH on track 1



(a) (b)

Values are normalized. High colour intensity (white) indicates a low frequency of usage while low colour intensity (black) indicates a high frequency of usage.

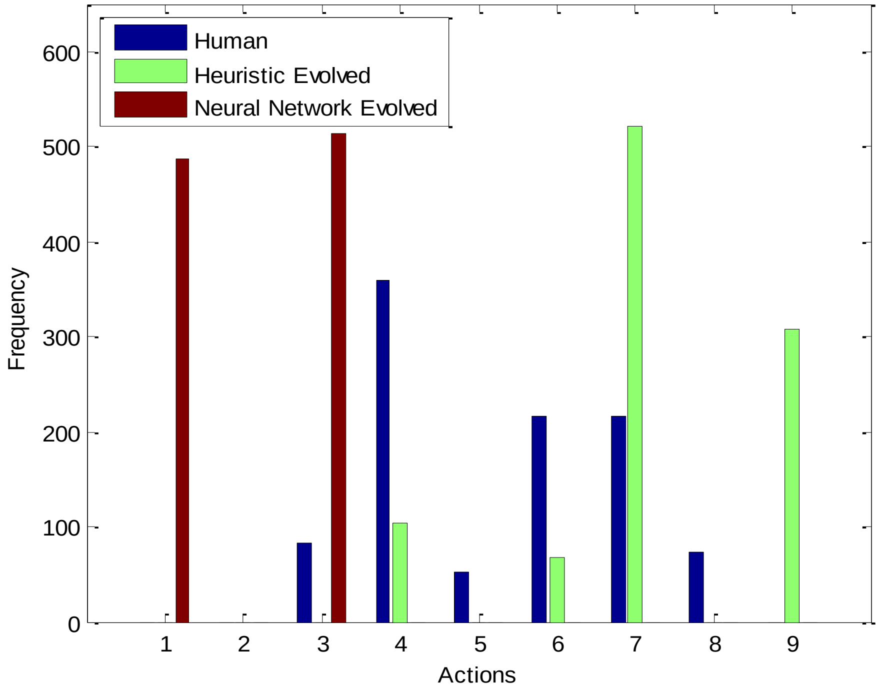
Figure 6.5 Histogram of the (a) output actions and (b) output action sequences of the ENN on track 1



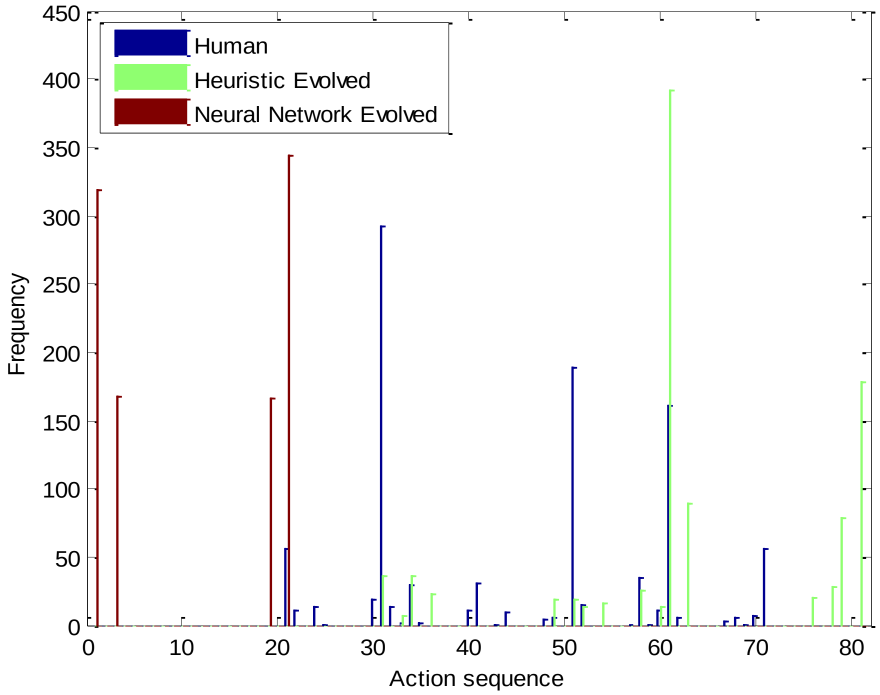
(a) (b)

Values are normalized. High colour intensity (white) indicates a low frequency of usage while low colour intensity (black) indicates a high frequency of usage.

Figure 6.6 Histogram of the (a) output actions and (b) output action sequences of the Hu on track 1



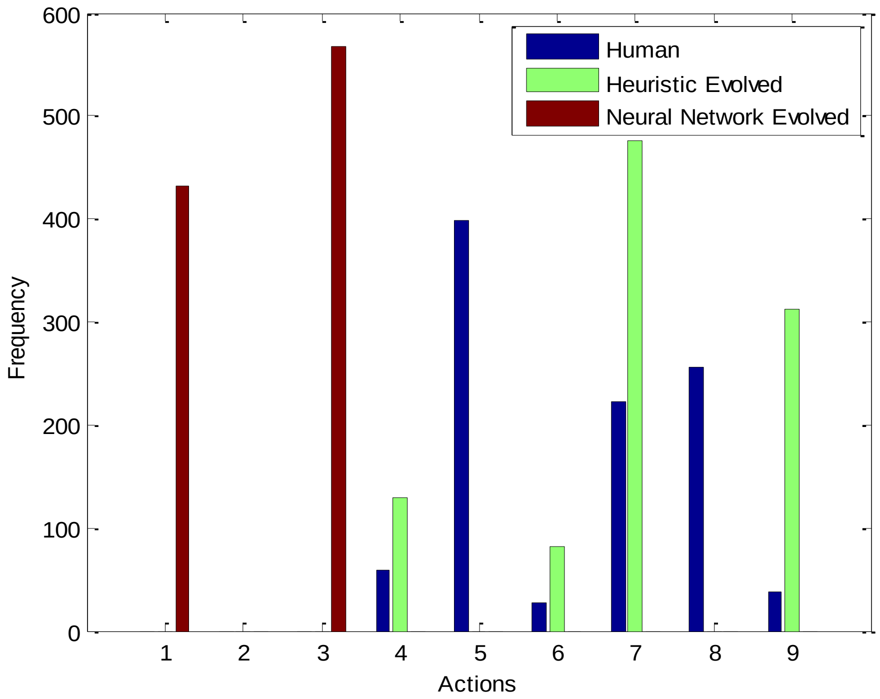
(a)



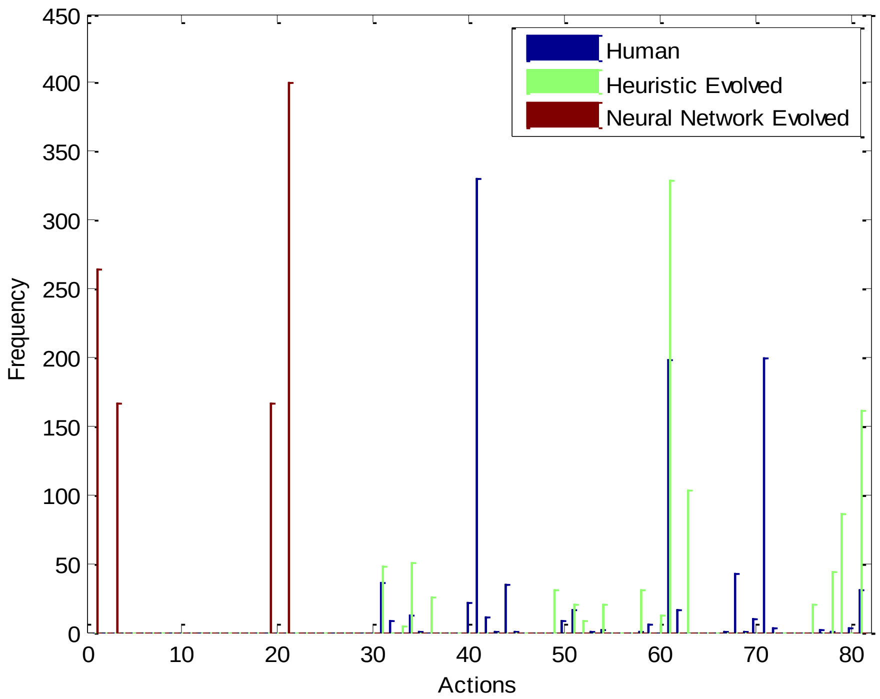
(b)

Action labeled on the horizontal axis (a) follows that in Table 6.1. Action sequence labeled on the horizontal axis (b) is derived from the equation 9a-9+b where an action sequence transits from action a to action b.

Figure 6.7 Comparative (a) action histograms and (b) action sequence histograms of human driving data, heuristic evolved controller, and neural network evolved controller on track 1



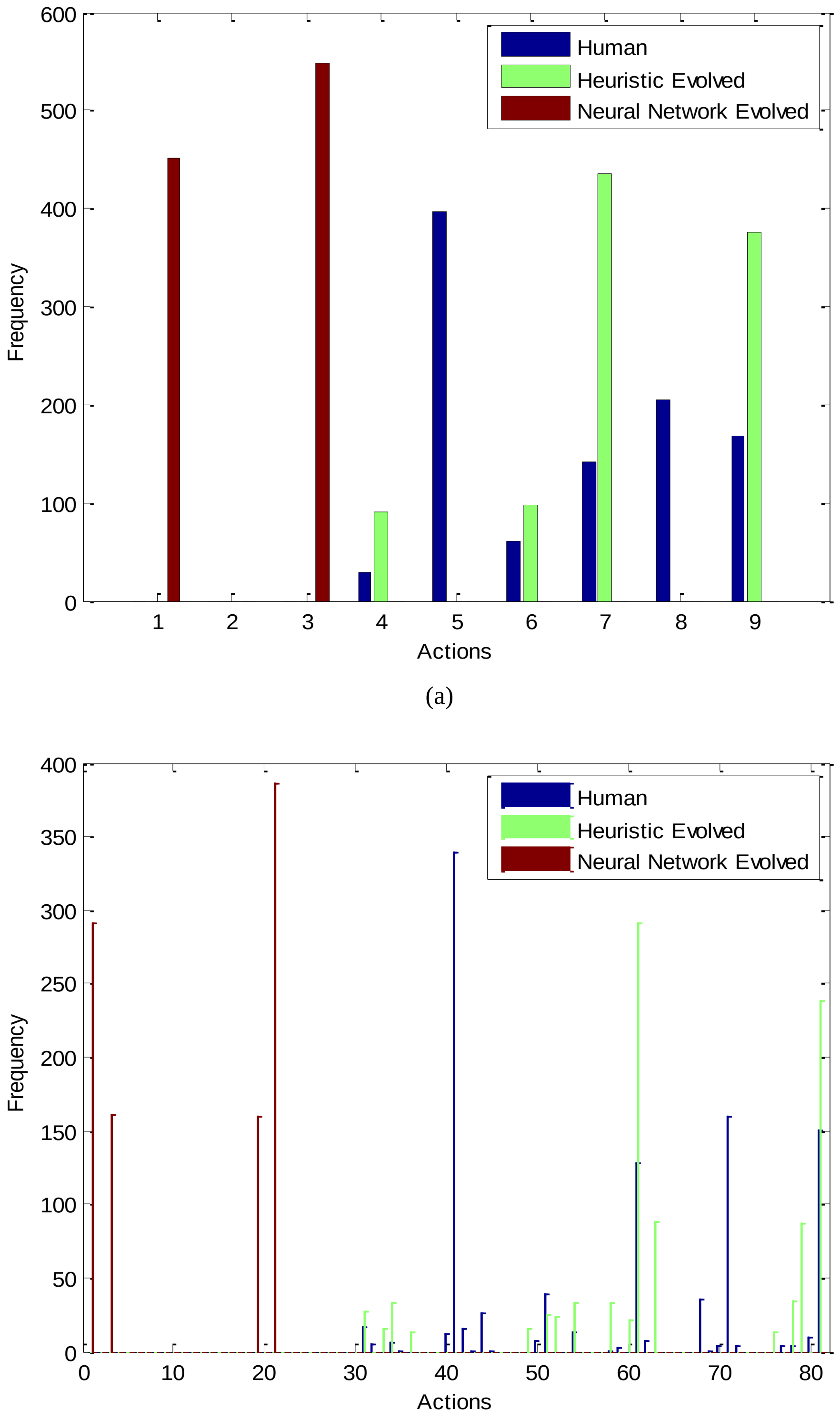
(a)



(b)

Action labeled on the horizontal axis (a) follows that in Table 6.1. Action sequence labeled on the horizontal axis (b) is derived from the equation 9a-9+b where an action sequence transits from action a to action b.

Figure 6.8 Comparative (a) action histograms and (b) action sequence histograms of human driving data, heuristic evolved controller, and neural network evolved controller on track 2

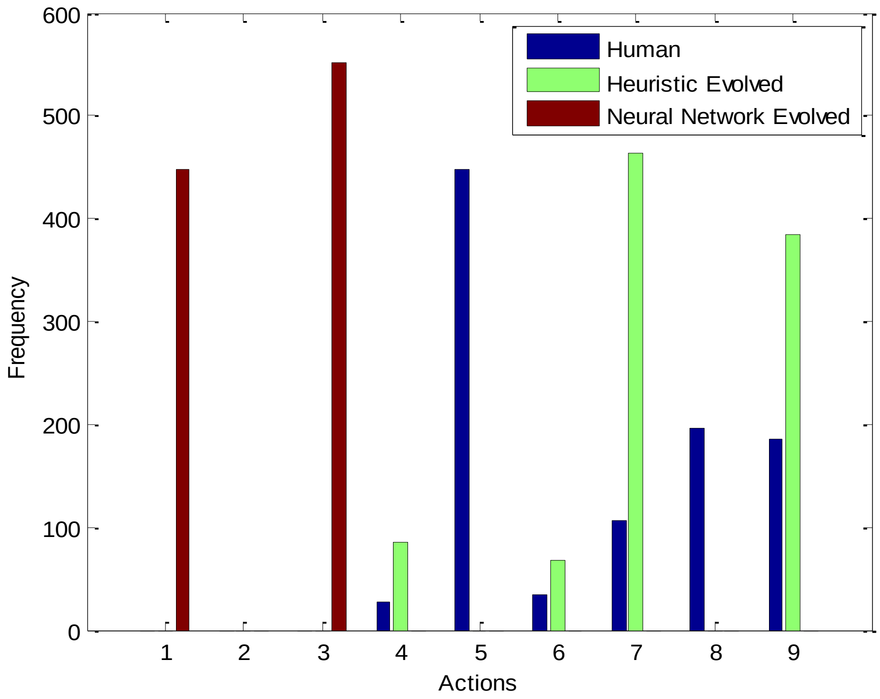


(a)

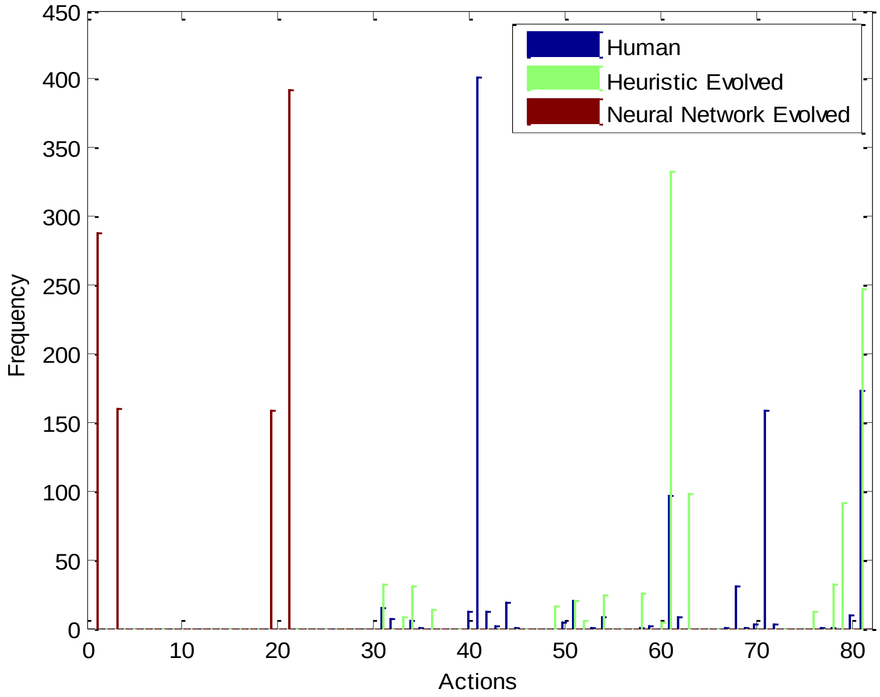
(b)

Action labeled on the horizontal axis (a) follows that in Table 6.1. Action sequence labeled on the horizontal axis (b) is derived from the equation 9a-9+b where an action sequence transits from action a to action b.

Figure 6.9 Comparative (a) action histograms and (b) action sequence histograms of human driving data, heuristic evolved controller, and neural network evolved controller on track 3



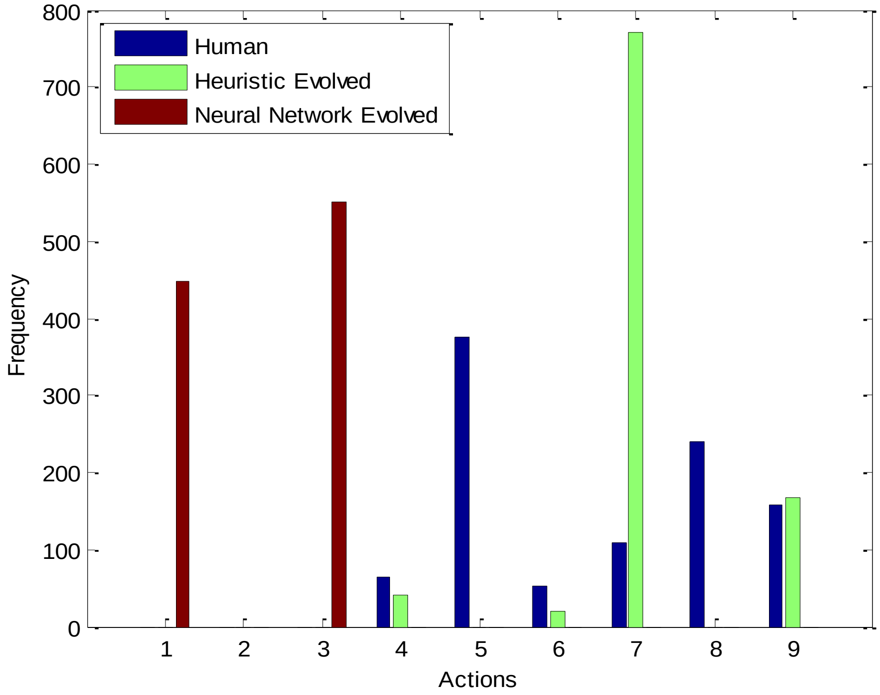
(a)



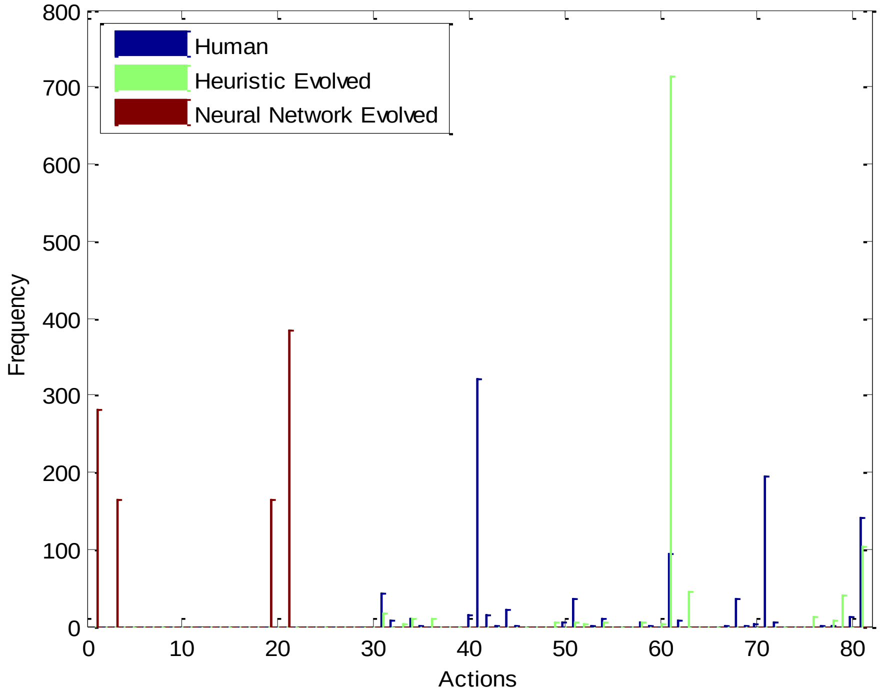
(b)

Action labeled on the horizontal axis (a) follows that in Table 6.1. Action sequence labeled on the horizontal axis (b) is derived from the equation 9a-9+b where an action sequence transits from action a to action b.

Figure 6.10 Comparative (a) action histograms and (b) action sequence histograms of human driving data, heuristic evolved controller, and neural network evolved controller on track 4



(a)



(b)

Action labeled on the horizontal axis (a) follows that in Table 6.1. Action sequence labeled on the horizontal axis (b) is derived from the equation 9a-9+b where an action sequence transits from action a to action b.

Figure 6.11 Comparative (a) action histograms and (b) action sequence histograms of human driving data, heuristic evolved controller, and neural network evolved controller on track 5

In the case of ENN, it was observed in Figure 6.5 (a) that the controller used only the actions 1 (Reverse Left) and 3 (Reverse Right). This meant that ENN was driving in reverse throughout the simulation. ENN found this as a better solution than driving forwards because the acceleration associated with reverse driving is smaller in magnitude compared to the acceleration for forward driving. The result of this was that ENN can better manipulate the driving trajectory of the car and avoid being trapped in orbits around waypoints due to large minimum turning radius. However, it was highly unnatural for a human player to drive a vehicle in reverse throughout the entire game. Moreover, ENN suffers the same constant steering but never neutral problem as EH.

The action histogram of Hu is shown in Figure 6.6 (a). It was observed that Hu used the actions 4 (Left), 5 (Neutral), 6 (Right), 7 (Forward Left), 8 (Forward) and 9 (Forward Right), and predominantly action 5. Hu did not drive in reverse because forward driving was more intuitive. Hu also spent a high percentage of the time doing nothing (i.e. action 5). This was to allow the car to slow down due to friction and also to observe feedback of the effects of its actions during previous time steps before making the next action. Additionally, action 8 (i.e. Forward only and neutral steering) was frequently used once Hu had aligned the car to the waypoint.

Next, the histogram of the action sequence of the controllers will be examined. It was observed in Figure 6.4 (b) that EH frequently used the sequences 7-7 (Forward Left-Forward Left), 7-9 (Forward Left-Forward Right), 9-7 (Forward Right-Forward Left) and 9-9 (Forward Right-Forward Right). The sequences 7-7 and 9-9 were natural as it implied that the controller needed to make a hard left turn (7-7) or hard right turn (9-9) and so it used the same action sequentially. However, it was unnaturally to frequently switch between right and left (7-9 and 9-7). During visual observations, EH could be seen oscillating its heading left and right about the straight line from its current position to the waypoint. This constant ‘fidgeting’ made its driving behaviour unnatural to the human observer.

The action sequence histogram of ENN is shown in Figure 6.5 (b). It was observed that ENN used only the sequences 1-1 (Reverse Left-Reverse Left), 1-3 (Reverse Left-Reverse Right), 3-1 (Reverse Right-Reverse Left) and 3-3 (Reverse Right-Reverse Right). Similar to that of EH, the sequences 1-1 and 3-3 were natural driving behaviours. However, ENN also had the same unnatural driving behaviour as with EH. It also oscillated its heading about the straight line from its current position to the waypoint, only this time in the reverse direction.

For Hu, it was observed from Figure 6.6 (b) that the more prominent sequences used were 4-4 (Left-Left), 5-5 (Neutral-Neutral), 6-6 (Right-Right), 7-7 (Forward Left-Forward Left), 8-8 (Forward-Forward) and 9-9 (Forward Right-Forward Right). In effect, Hu frequently repeated its actions and seldom switched to other actions. In contrast to EH and ENN, Hu did not frequently use left to right or right to left switching. The result was a more believable driving behaviour.

The discussion above can easily be generalized to other tracks as demonstrated by the similarity of the histograms in Figure 6.7, Figure 6.8, Figure 6.9, Figure 6.10 and Figure 6.11. The lack of neutral commands was evident in all tracks and the differences in frequencies of left and right commands was a result of the track profile.

From the observations above, the advantages and the disadvantages of evolutionary computation were demonstrated. The evolved controllers were able to exploit the fitness function to find good driving controllers that maximize the number of waypoints passed. They were also able to find unexpected solutions that satisfied the fitness function just as well. However, unexpected solutions can be a double edged sword. In data mining problems, unexpected solutions can lead to the discovery of novel relationships amongst large data sets. But in application to gaming, unexpected solutions can ruin the suspension of disbelief for the user, thereby reducing their satisfaction in the game. As shown in this case study, the evolved controllers produced unexpected and also unnatural driving behaviours.

In this section, the action histograms and action sequence histograms of previously evolved controllers were quantitatively and qualitatively analyzed and associated to some of the unnatural and unrealistic driving behaviours that were visually observed. It could be seen that the differences in driving behaviours between the evolved controllers and the human player could be traced to the types of actions and sequence of actions used during the simulation.

As such, the use of action histograms and action sequence histograms is proposed as a form of guided training so as to evolve believable controllers that appear more natural to human players by imitating the low level behavioural tendencies of human players. That is, if the evolved controller is able to learn the histograms of the data collected from human driving, then the evolved controller will drive in a more believable manner. As driving competency and driving believability are not directly related, the basic single objective evolutionary framework will be unable to optimize both these criteria at the same time. Therefore, the multi-objective evolutionary framework needs to be introduced to cope with the addition of believability as a second objective.

6.3.5 Histograms of small window sizes

Following the above discussion, it can be seen that histograms of larger window sizes up to n is possible. The number of input samples in a histogram is related to the window size by equation (6.9).

$$k = n - \left(w - 1 \right) \tag{6.9}$$

where k is the number of input samples in a histogram of window size w, and n is the total number of observations. In the case where n = w, there would only be one input to the histogram.

The size of a histogram (i.e. the number of frequency bins) of window size w is given by 9^{w}. It was observed that with increasing window size, the number of input samples decreased linearly while the number of frequency bins in the corresponding histogram increased exponentially. This would result in many unfilled frequency bins. Unfilled bins are undesirable because an evolutionary algorithm will not be able to distinguish one unfilled bin from another. That is, unfilled bins do not provide useful information to guide the evolution. Therefore, increasing the window size will make the fitness landscape increasingly complex and difficult. To illustrate, if a histogram of window size n (i.e. n = w = 1000) is used, then only one bin will be filled while all the other $9^w - 1$ bins will be unfilled. This encourages the evolution process to find a solution that will imitate the training data exactly. Not only is this problem difficult, it will also likely produce a solution that will not be able to generalize well to other unseen situations. This makes histograms of larger window sizes not suitable as candidates for fitness functions.

The choice of using histograms of small window sizes as fitness functions serves yet another objective. The objective of this experiment is not to imitate human behaviours. Rather, the aim is to induce non-player characters (NPC) in a game with humanlike characteristics through evolution. That is, the window sizes of the histograms are deliberately made small in order to capture low level reactionary behavioural tendencies in humans rather than high level strategic planning. Strategies are problem dependent (i.e. track dependent) but reactionary behaviours tend to be consistent. For instance, characters may choose to smile or frown depending on who they are talking to, but they will always blink their eyes. Hence, the goal is to improve the believability of an NPC through the induction of such behavioural tendencies (i.e. learning how to blink our eyes). Additionally, the use of histograms of small window sizes discourages the learning of long action sequence chains during training so that the evolved solution is more likely to generalize better to situations other than the ones in training. The proposed action histograms framework is designed to work with other games of similar discrete control schemes such as platform games (i.e. Super Mario, Rayman) and arcade shooters (i.e. Asteroids, Space Invaders).

6.4 *Fitness functions*

The controller needs to drive well on a given track and at the same time drive in a believable manner. Hence, the fitness functions to use for evolution must be able to guide the evolution of the controller towards a balance of these incomparable objectives.

6.4.1 Waypoints

The number of waypoints is used as an objective to evolve driving performance. How well a controller drives on a given track can be directly measure by the distance covered by the car within a stipulated time. In the experiments, the time of each game is fixed at 1000 time steps. Each track is defined by the sequential order of its waypoints within a square, obstacle-free game area. From one waypoint to the next, a controller is not confined to any particular path. In practice, the controller may choose to drive around a large circular path or simply a straight line towards the next waypoint. Hence, directly measuring the distance covered by the car may not be a good indicator of the driving ability of a controller in a track. Instead, the number of waypoints passed by a car is used as a measure of the racing ability of a controller. That is, the more waypoints passed the more effective the controller is in driving towards its destination. Therefore, the objective is to maximize the number of waypoints passed.

6.4.2 Histo1 (Action histogram)

The action histogram is the first of two fitness functions used as an objective to evolve believability. An evolved controller is considered to be believable if it is able to drive around a given training track using a set of actions similar to that of a human player driving on the same track. For a given training track, the objective is to minimize the sum of squared difference between the action histogram of the evolved controller and that of the human player, as given in equation (6.10).

$$\min f_1 = \sum_{i=1}^{9} \left(H_i - m_i \right)^2 \tag{6.10}$$

where H_i is the number of observations that fall into bin i for the human player and m_i is the number of observations that fall into bin i for the evolved controller.

6.4.3 Histo2 (Action sequence histogram)

The action sequence histogram is the second function used as an objective to evolve believability. The effectiveness of both the action histogram and the action sequence histogram will be compared in subsequent experiments. An evolved controller is considered to be believable if it is able to drive around a given training track using a set of actions transitions similar to that of a human player driving on the same track. For a given training track, the objective is to minimize the sum of squared difference between the action sequence histogram of the evolved controller and that of the human player, as given in (6.11).

$$\min f_2 = \sum_{i=1}^{81} \left(H_i - m_i \right)^2 \tag{6.11}$$

where H_i is the number of observations that fall into bin i for the human player and m_i is the number of observations that fall into bin i for the evolved controller.

6.5 Single objective evolution

The purpose of the single objective experiments is to demonstrate the incomparable nature of the good driving and believable driving. In addition, the effects of introducing sensor noise will also be discussed. In the single objective experiments, a (40+40) ES [124], running for 200 generations was used as a training method. Self adaptive learning was not applied. The mutation operator was a Gaussian perturbation with a step size of 0.1 and a probability of 0.9. Tournament selection was used and elitism was set to 10%. Each individual was evaluated on its own (i.e. solo game) and the results were averaged over 10 evaluations. Two fitness functions were compared, the number of waypoints and the sum of squared errors (SSE) of Histo1. Each set of experiments was repeated with sensor noise and without sensor noise. In the case without sensor noise, each individual was encoded with 10 real value variables. In the case with sensor noise, each individual was encoded with 18 real value variables. The track used for training was track 1.

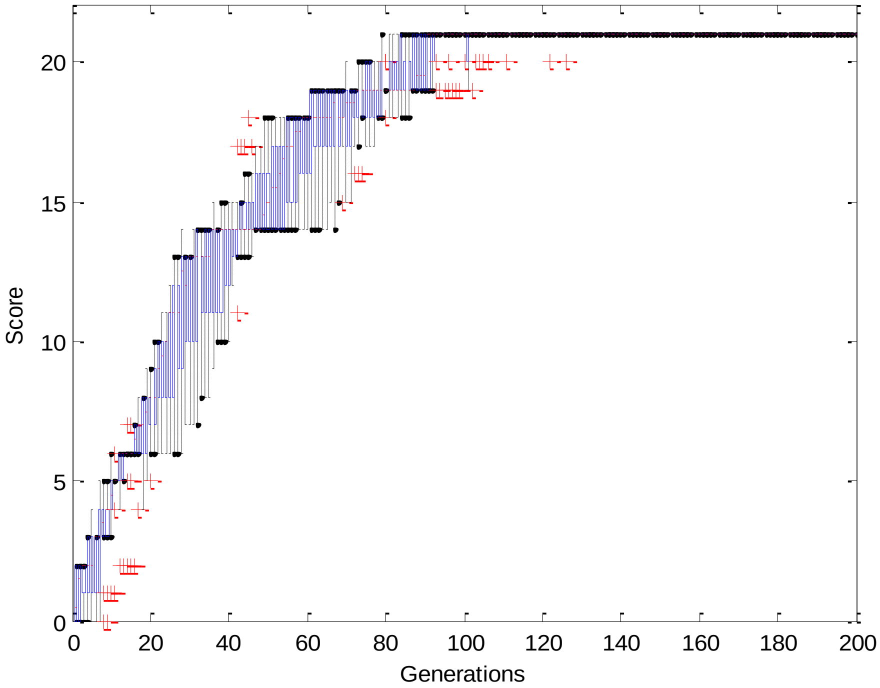
6.5.1 Number of waypoints

The objective was to maximize the number of waypoints passed. The experiment was conducted for 2 cases, without sensor noise and with sensor noise. For each experiment, both the number of waypoints and the SSE of Histo1 are presented for discussion although only the number of waypoints was used as the fitness function.

6.5.1.1 Without noise

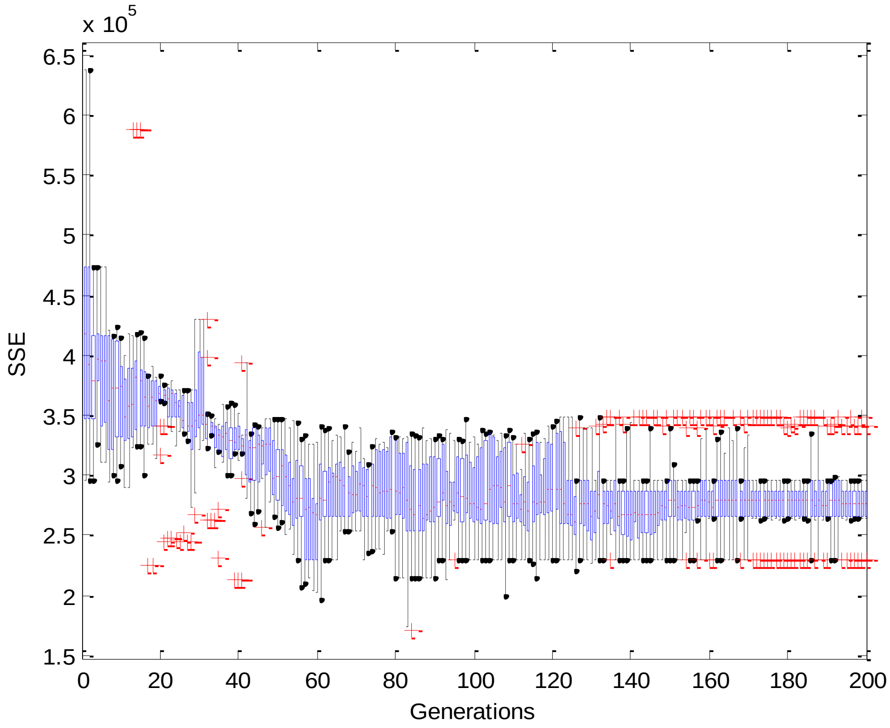
The training fitness for the case without sensor noise is presented in Figure 6.12 and Figure 6.13. The figures were obtained by plotting the boxplot of the fitness of the best individual at each generation of 10 independent runs.

It was observed from Figure 6.12 that the fitness of independent runs converged to a score of 21 waypoints at 92 generations. The score of 21 waypoints was also significantly higher than the 17.8 waypoints for human driving. Next, the values of the SSE of Histo1 over the generations were examined to see if there were observable relationships to the number of waypoints. It was observed from Figure 6.13 that the SSE steadily decreased over the first 60 generations. This result was expected as the controllers evolved from random actions to directed actions that drove towards the waypoints.



Boxplot of the number of waypoints at every generation up to 200 generations.

Figure 6.12 Boxplot of the number of waypoints for single objective optimization to maximize number of waypoints, without sensor noise



Boxplot of the sum of square errors of Histo1 (actions) at every generation up to 200 generations. Final mean value is 2.790×10^5.

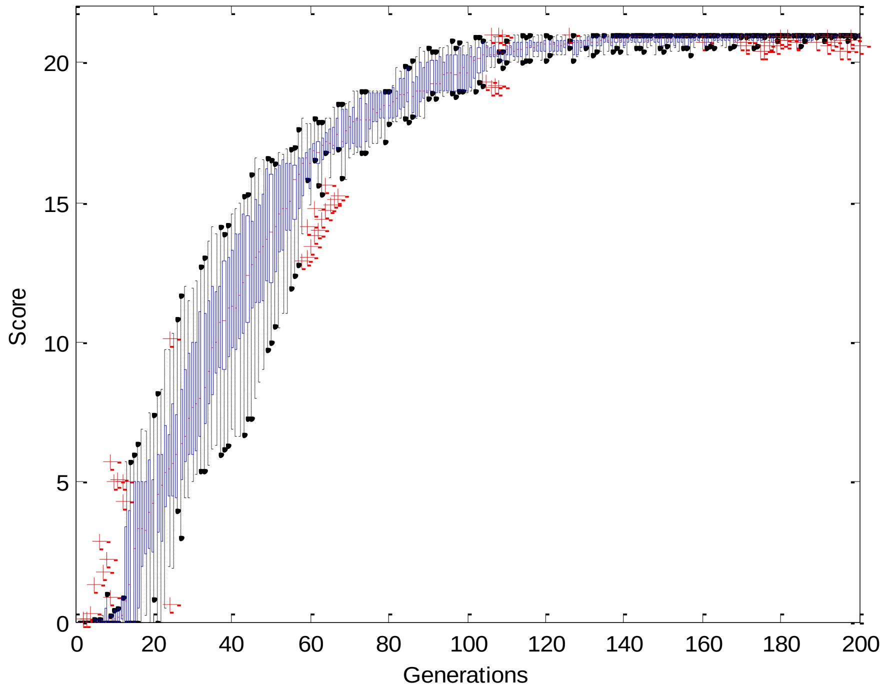
Figure 6.13 Boxplot of the sum of square errors of Histo1 for single objective optimization to maximize number of waypoints, without sensor noise

However, the SSE of Histo1 stagnated after 60 generations to a mean of 2.790×10^5 even as the number of waypoints continued to increase. Visual observations of the evolved controllers revealed that the evolved behaviour was identical to that of the Evolved Heuristic Controller. This indicated that the number of waypoints scored and SSE of Histo1 were not directly related.

6.5.1.2 With noise

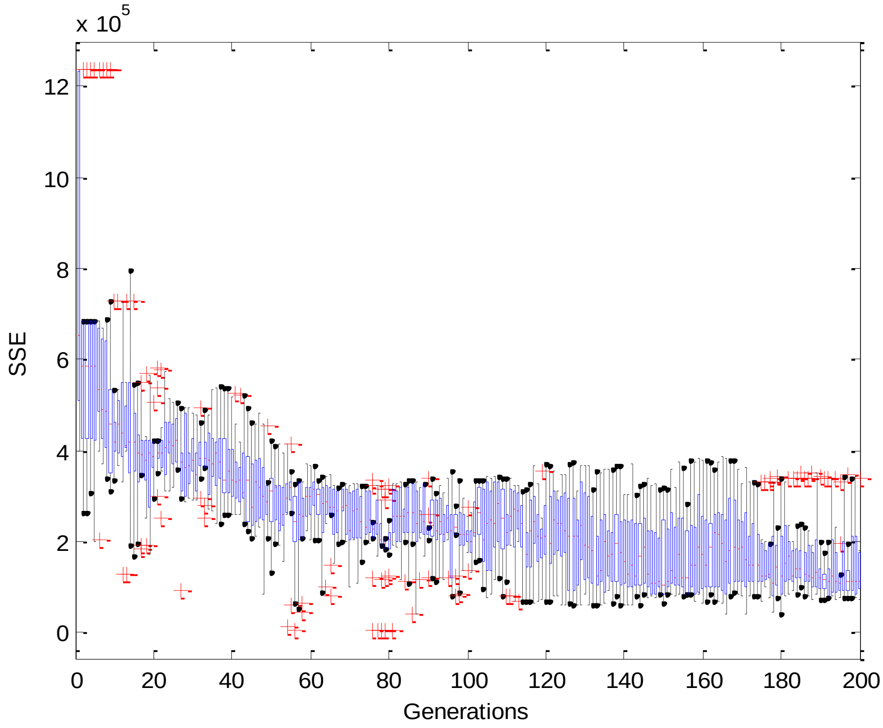
Next, sensor noise was introduced to the controller to investigate if the use of noisy sensors to imitate errors in human judgments would improve the believability of the evolved controllers. The training fitness for the case with sensor noise is presented in Figure 6.14 and Figure 6.15.

In Figure 6.14, it was observed that the population converged to the same optimal waypoint score of 21, similar to the case without the introduction of sensor noise. Therefore, it could be said that the introduction of sensor noise did not degrade the driving performance of the evolved controller, and that evolving the sensor noise parameters and the controller parameters together was feasible. However, it did delay the rate of convergence as the population converged after about 120 generations compared to 92 generations without noise. This result was expected as there were more variables to evolve in the case with noise. Furthermore, the stochastic nature of the sensors made the search space more complicated.



Boxplot of the number of waypoints at every generation up to 200 generations.

Figure 6.14 Boxplot of the number of waypoints for single objective optimization to maximize number of waypoints, with sensor noise



Boxplot of the sum of square errors of Histo1 (actions) at every generation up to 200 generations. Final mean value is 1.381×10^5.

Figure 6.15 Boxplot of the sum of square errors of Histo1 for single objective optimization to maximize number of waypoints, with sensor noise

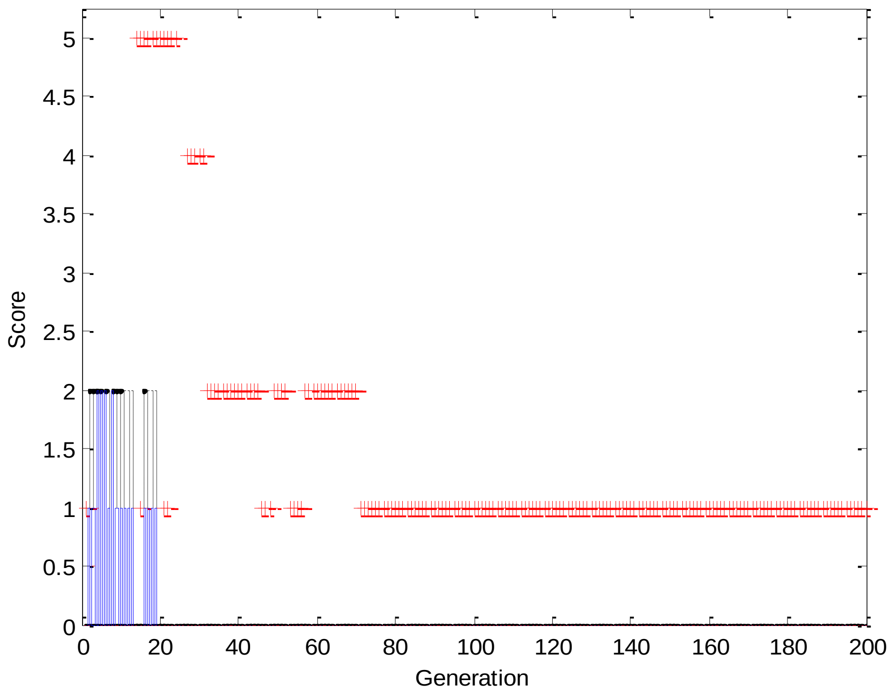
It was observed from Figure 6.15 that the SSE of Histo1 was decreasing throughout the evolution, even after the waypoint score converged to a mean of 1.381×10^5 after 120 generations. The value of the SSE was also lower than the case without noise (i.e. 2.790×10^5). Firstly, this showed that for the same performance in waypoint score, the SSE of Histo1 had the potential to be further reduced. Secondly, the introduction of sensor noise to the evolved controllers had the potential to be effective in improving the believability of the evolved controllers (i.e. reduce SSE of Histo1).

6.5.2 Action histogram (Histo1)

In this section, the objective was changed to minimize the SSE of the Histo1. The training track and data used was track 1. The experiment was conducted for 2 cases, without sensor noise and with sensor noise. For each experiment, both the number of waypoints passed and the SSE of Histo1 are presented for discussion although only the SSE of Histo1 was used as the fitness function.

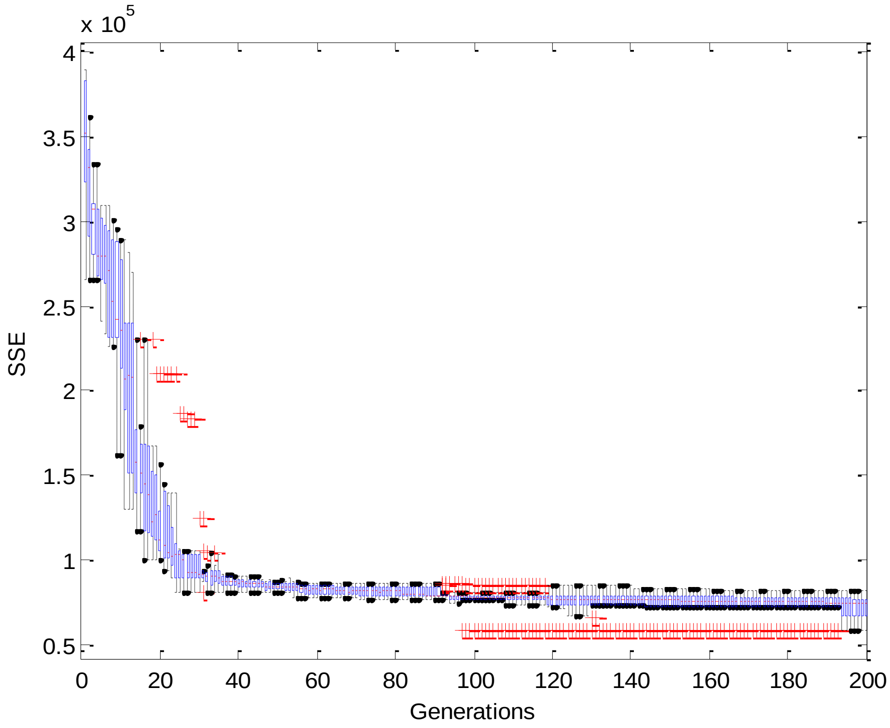
6.5.2.1 Without noise

The training fitness for the case without sensor noise is presented in Figure 6.16 and Figure 6.17. The figures were obtained by plotting the respective boxplot of the fitness of the best individual at each generation of 10 independent runs.



Boxplot of the number of waypoints at every generation up to 200 generations.

Figure 6.16 Boxplot of the number of waypoints for single objective optimization to minimize the sum of squared errors of Histo1, without sensor noise



Boxplot of the sum of squared errors of Histo1 (actions) at every generation up to 200 generations. Final mean value is 7.165×10^4.

Figure 6.17 Boxplot of the sum of squared errors of Histo1 for single objective optimization to minimize the sum of squared errors of Histo1, without sensor noise

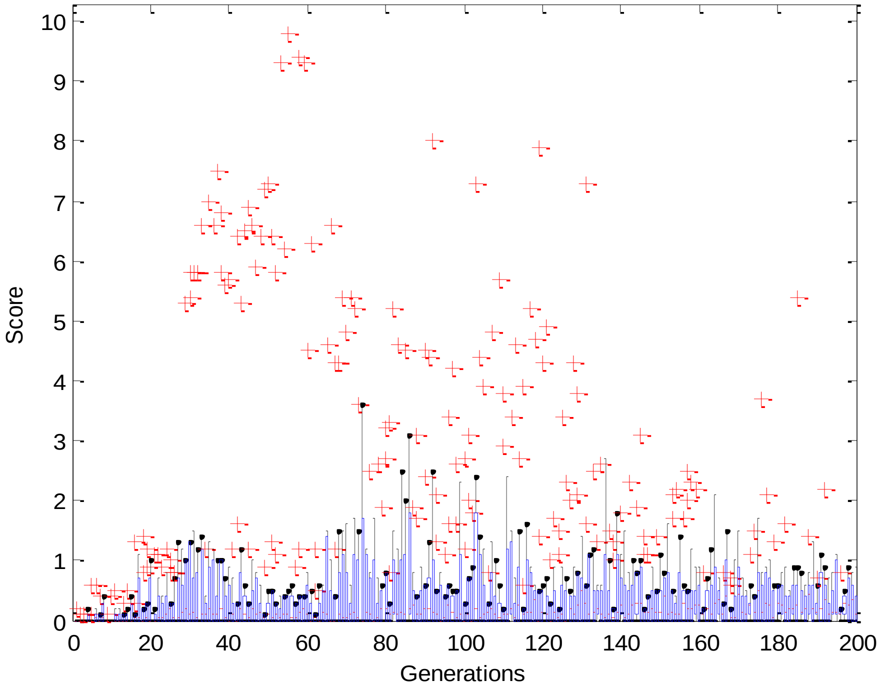
It was observed from Figure 6.16 that the waypoint score rose to as high as 5 points, albeit only an outlier, in the first 30 generations although it was not used as a fitness function. This was likely due to the random movement of the evolved controller that coincidentally passed through some waypoints. For the remaining of the generations, the waypoint score remained at zero.

In Figure 6.17, it was observed that the SSE of Histo1 decreased rapidly for the first 30 generations. In the remaining generations, only incremental improvements were observed. The final value of the SSE of Histo1 had a mean of 7.165×10^4 which was lower than the case of optimizing only waypoint score with sensor noise (i.e. 1.381×10^5). This implied that the SSE could still be further reduced.

Taking both Figure 6.16 and Figure 6.17 as a whole, it was observed that optimizing Histo1 alone did not improve driving performance. This result further reinforced that the number of waypoints scored and the SSE of Histo1 were not directly related. Hence, a multi-objective framework was necessary to optimize both objectives simultaneously.

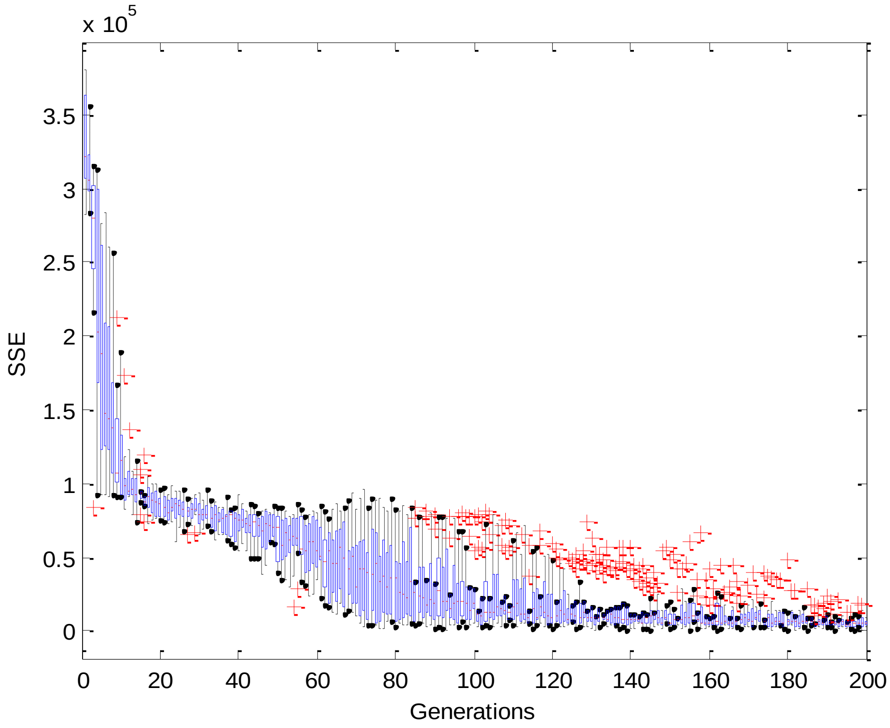
6.5.2.2 With noise

Next, sensor noise was introduced to the controllers and evolved using the SSE of Histo1 as the fitness function. The training fitness for the case with sensor noise is presented in Figure 6.18 and Figure 6.19.



Boxplot of the number of waypoints at every generation up to 200 generations.

Figure 6.18 Boxplot of the number of waypoints for single objective optimization to minimize the sum of squared errors of Histo1, with sensor noise



Boxplot of the sum of squared errors of Histo1 (actions) at every generation up to 200 generations. Final mean value is 5.486×10^3.

Figure 6.19 Boxplot of the sum of squared errors of Histo1 for single objective optimization to minimize the sum of squared errors of Histo1, with sensor noise

It was observed in Figure 6.18 that the waypoint scores between independent runs during training had a higher standard deviation compared to the case without sensor noise. This was due to the stochastic nature of the noise in the sensors which resulted in more coincidental passing through of waypoints. However, the scores were still close to zero and no upward trend was observed.

From Figure 6.19, it was observed that the SSE of Histo1 reduced rapidly for the first 70 generations before converging. The mean value of the converged SSE of Histo1 was 5.486×10^3, significantly lower than 7.165×10^4 in the case of optimizing the SSE of Histo1 without noise, 1.381×10^5 in the case of optimizing waypoint score with noise, and 2.790×10^5 in the case of optimizing waypoint score without noise. This meant that there was, on average, an error of 24.69 actions per bin in Histo1 for optimizing the SSE of

Histo1 with noise compared to an average of 176.1 error actions per bin in the case of optimizing waypoints without noise. This result was significant because it demonstrated that evolving controllers to imitate human behaviours using action histograms as the fitness function was feasible. The range of acceptable values for the SSE of Histo1 should be in the magnitude of 10^3. The next step would involve evolving controllers that were able to drive well and believably at the same time using the multi-objective framework.

6.6 Multi-objective evolution

In the previous section, it was demonstrated that the objectives of driving well (i.e. number of waypoints) and driving believably (i.e. SSE of Histo1) were not directly related. In the single objective experiments, either one of the objectives could be optimized but not both simultaneously. It was also demonstrated that the inclusion of sensor noise that evolved together with the controllers was feasible and could improve the SSE of Histo1 without degrading the driving performance of the controller. In this section, both driving performance and driving believability will be optimized together using the multi-objective (MO) evolutionary framework. Multi-objective optimization is introduced in this work to find a balance between two incomparable objectives, driving performance and driving believability. That is, the controller needs to be able to drive well on a given track and at the same time drive in a believable manner.

The experiments conducted in this section will be discussed in four parts. The training results are discussed first. Next, the effects of the evolved sensor noise are examined. The generalization capability of the evolved controllers is covered in the third section. Finally, a user study is conducted and its results are analyzed.

In order to be more concise, the controllers of interest evolved during training will be abbreviated according to the list given in Table 6.4. Other frequently used controllers are also abbreviated.

Table 6.4 Abbreviated list of controllers that are frequently used in text

Controller name	Objective 1	Objective 2	Remarks
H1H	Waypoints	Histo1	High score
H1L			Low SSE
H2H	Waypoints	Histo2	High score
H2L			Low SSE
Hu	-	-	Human
EH	Waypoints	-	Heuristic [166]
ENN	Waypoints	-	Neural network [166]

6.6.1 Training

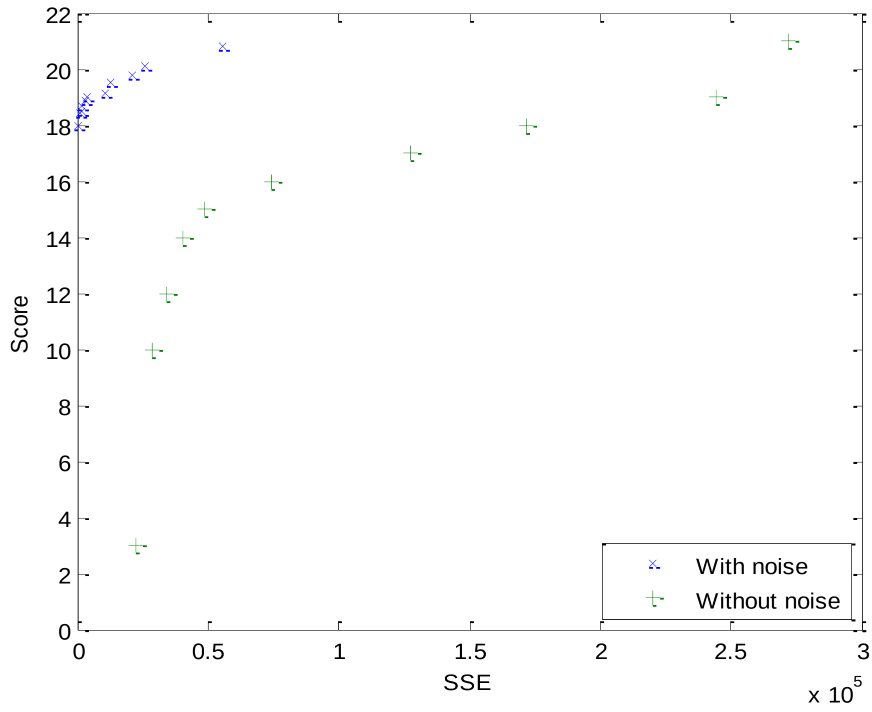
In this section, both driving performance and driving believability will be optimized simultaneously using the MO evolutionary framework. The parameters used in the MO experiments were identical to that of the single objective experiments. A (40+40) ES, running for 200 generations was used as a training method. Self adaptive learning was not applied. The mutation operator was a Gaussian perturbation with a step size of 0.1 and a probability of 0.9. Each individual was evaluated on its own (i.e. solo game) and the results averaged over 10 evaluations. Two combinations of fitness functions were considered; first, maximize number of waypoints and minimize the SSE of Histo1, and second, maximize the number of waypoints and minimize the SSE of Histo2. At each generation, the individuals were ranked in terms of Pareto optimality, tournament selection was used. The objective was to compare the differences and effectiveness of using Histo1 or Histo2 to evolve believable behaviours. Each set of experiments was repeated without sensor noise and with sensor noise. In the case without sensor noise, each individual was encoded with 10 real value variables. In the case with sensor noise, each individual was encoded with 18 real value variables. The track used for training was track 1. Note that because Histo1 consists of 9 frequency bins while Histo2 consists of 81 frequency bins, and Histo1 is populated by 1000 actions samples while Histo2 is populated by 999 action sequences samples, the SSE values of Histo1 and Histo2 cannot be directly compared.

6.6.1.1 Waypoints and Histo1 (action histogram)

In this experiment, the objective was to maximize the number of waypoints scored and minimize the SSE of Histo1. The experiment was repeated without sensor noise and with sensor noise. The results were obtained from 10 independent runs and the non-dominated controllers are plotted in Figure 6.20.

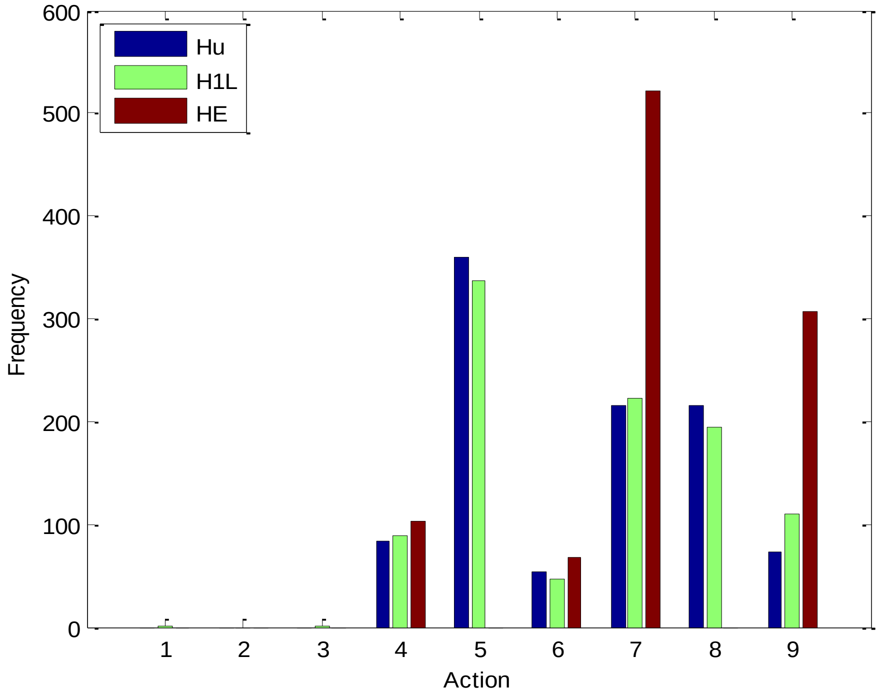
For both experiments, without and with noise, a clear Pareto front can be observed. These indicated that there existed a tradeoff between the number of waypoints scored and the SSE of Histo1. A high waypoint score could only be achieved by driving less humanlike, while a more humanlike driving behaviour would result in a lower waypoint score.

It was also observed that the Pareto front of the case with sensor noise dominated the Pareto front of the case without sensor noise. It showed that introducing sensor noise to simulate realistic human judgment errors was necessary and had the effect of evolving controllers that drive more believably without degrading its driving performance. More details about the effects of sensor noise will be discussed in the next section.



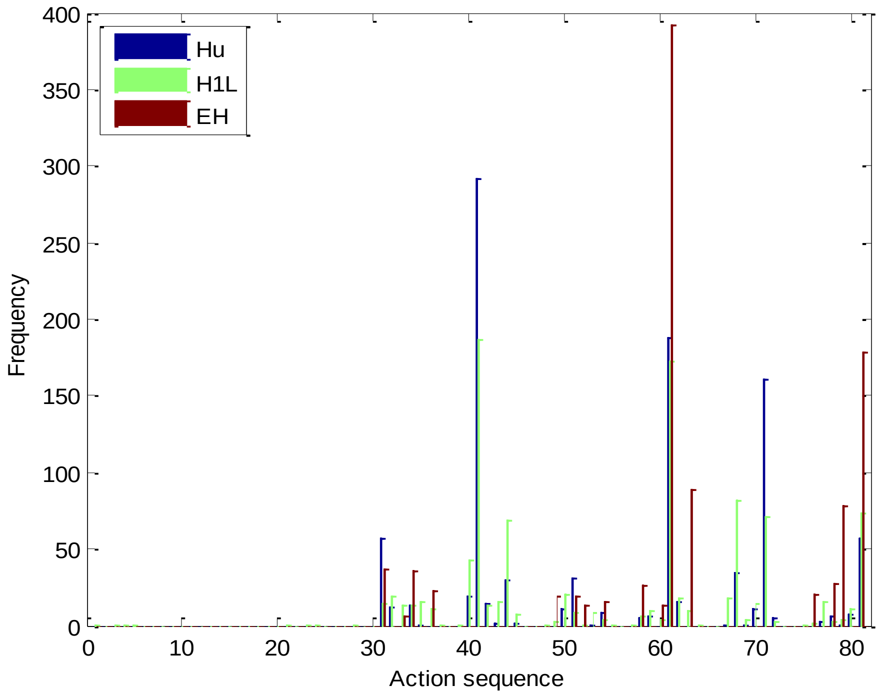
Results are plotted at the end of 200 generations for experiments with and without sensor noise. With noise, the data ranged from (502.9, 18) to (5.55×10^{4}, 20.8). Without noise, the data ranged from (2.215×10^{4}, 3) to (2.722×10^{5}, 21).

Figure 6.20 Multi-objective optimization to maximize the number of waypoints and minimize the sum of squared errors of Histo1



Action labeled on the horizontal axis follows that in Table 6.1.

Figure 6.21 Comparative action histograms of Hu, H1L, and EH (left to right)



Action sequence labeled on the horizontal axis is derived from the equation 9a-9+b where an action sequence transits from action a to action b.

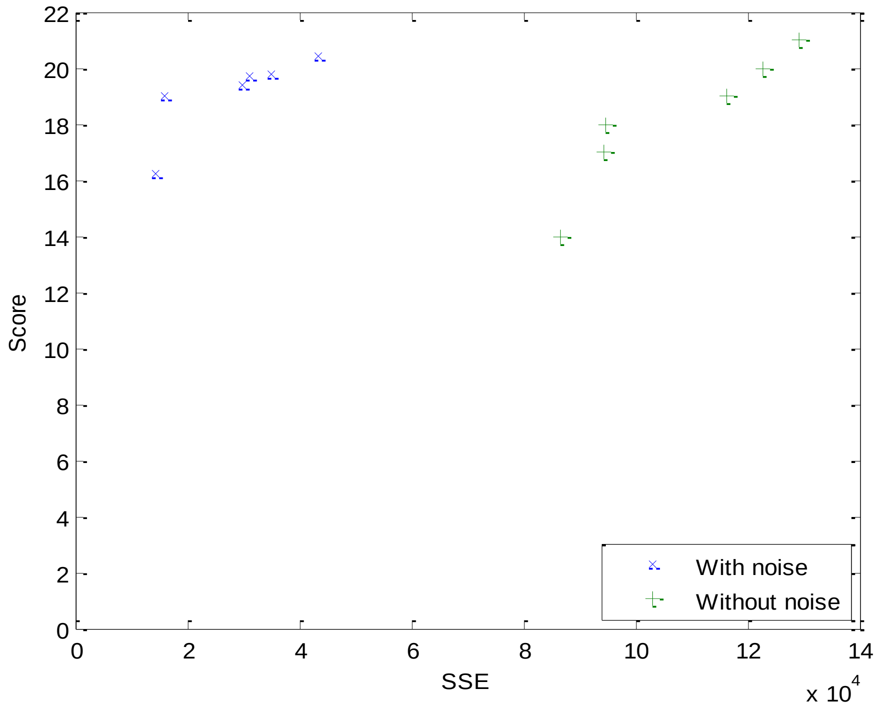
Figure 6.22 Comparative action sequence histograms of Hu, H1L, and EH (left to right)

The next consideration for designers was to select the most suitable controller from the set of candidate solutions in the Pareto front. In this study, two ways of choosing the solution controller will be recommended. First, choose the controller with the lowest SSE because it had an action histogram that most resemble human driving data. Or second, choose the controller with a waypoint score that best match the human driving score. This will provide a good tradeoff of driving performance and believability. The solution with the lowest SSE (H1L) was selected for the purpose of comparison. The histograms from Hu, H1L, and EH are presented in bar chart form in Figure 6.21 and Figure 6.22.

It was observed from Figure 6.21 that H1L was able to drive using actions similar to Hu terms of Histo1. There was a significant reduction in actions 7 and 9 compared to EH. Also, H1L made frequent use of actions 5 and 8 which were used in Hu but were not used by EH. In Figure 6.22, it was observed that H1L was also able to match Hu better than EH in terms of Histo2.

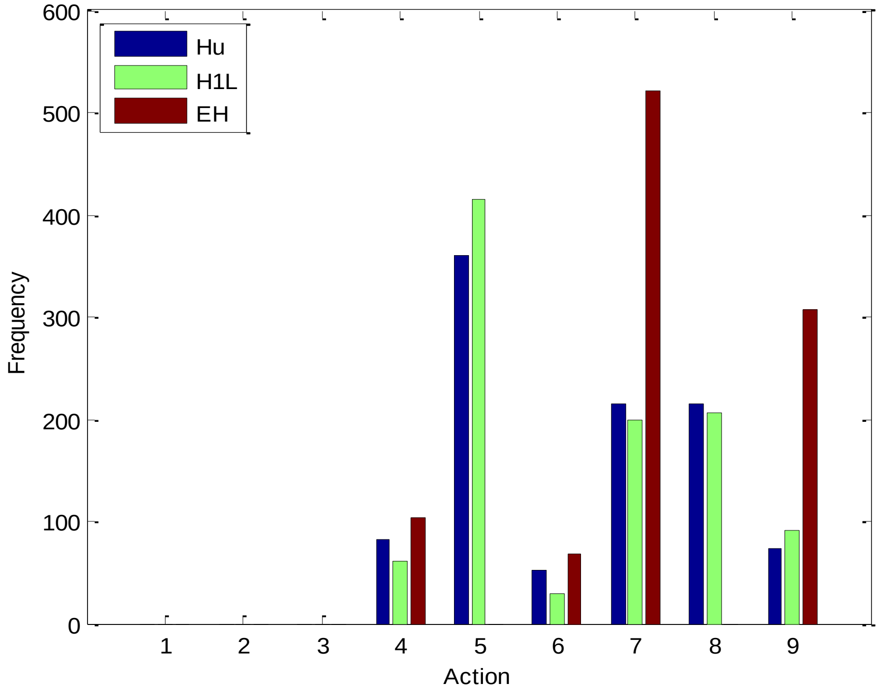
6.6.1.2 Waypoints and Histo2 (action sequence histogram)

In this experiment, the objective was to maximize the number of waypoints score and minimize the SSE of Histo2. This experiment was also repeated for cases without and with sensor noise. The results were obtained from 10 independent runs and the non-dominated controllers are plotted in Figure 6.23.



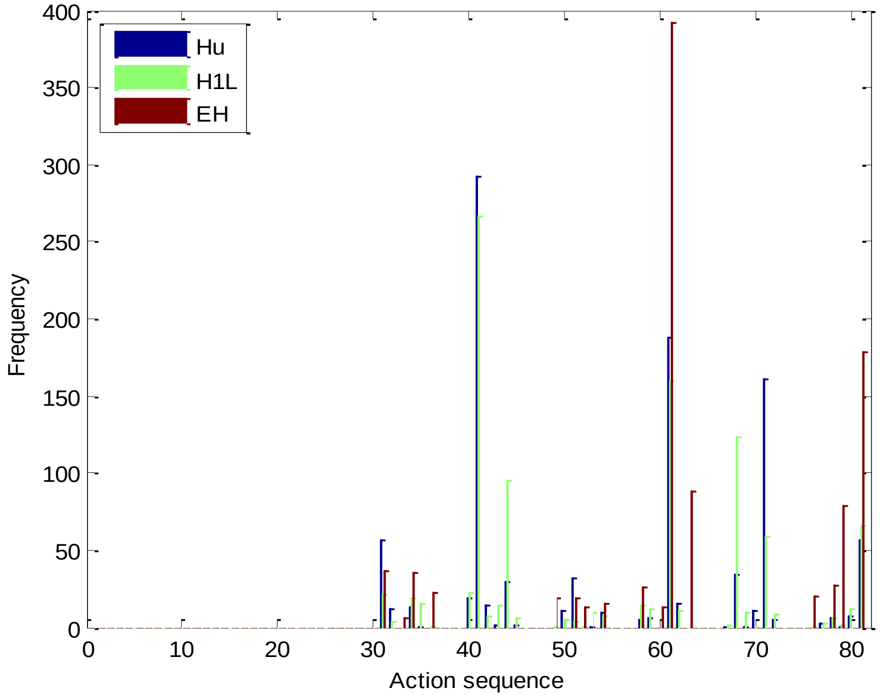
Results are plotted at the end of 200 generations for experiments with and without sensor noise. With noise, the data ranged from (1.432×10^4, 16.2) to (4.335×10^4, 20.4). Without noise, the data ranged from (8.673×10^4, 14) to (1.293×10^5, 21).

Figure 6.23 Multi-objective optimization to maximize the number of waypoints and minimize the sum of squared errors of Histo2



Action labeled on the horizontal axis follows that in Table 6.1.

Figure 6.24 Comparative action histograms of Hu, H2L, and EH (left to right)



Action sequence labeled on the horizontal axis is derived from the equation 9a-9+b where an action sequence transits from action a to action b.

Figure 6.25 Comparative action sequence histograms of Hu, H2L, and EH (left to right)

A point to note here was that the SSE of Histo1 and the SSE of Histo2 were not directly comparable and should not be viewed as such. This was because of the difference in histogram space. There were only 9 bins in Histo1 compared to 81 bins in Histo2. For example, assuming the 1000 actions to be evenly distributed in the 9 bins of Histo1, the SSE for track 1 would be 195397.6. Assuming the 999 action sequences to be evenly distributed in the 81 bins in Histo2, the SSE for track 1 would be 570787.3. Therefore, it was expected that the optimized values of SSE of Histo1 would be usually lower than the optimized values of SSE of Histo2 because there was a lesser number of frequency bins in Histo1.

It was observed that a clear Pareto front was created for both without and with sensor noise. This indicated that there existed a tradeoff between the number of waypoints scored and the SSE of Histo2. It was also noted that the controllers with sensor noise were able to form a Pareto front that dominated the one formed by the controllers without sensor noise. This reinforced the findings that introducing sensor noise to simulate human judgment errors had the effect of evolving controllers that drove more believably without degrading driving performance.

In this experiment, the controller with the lowest SSE of Histo2 (H2L) was selected for comparison with the data from controllers Hu and EH. The histograms of these controllers are presented in bar chart form in Figure 6.24 and Figure 6.25.

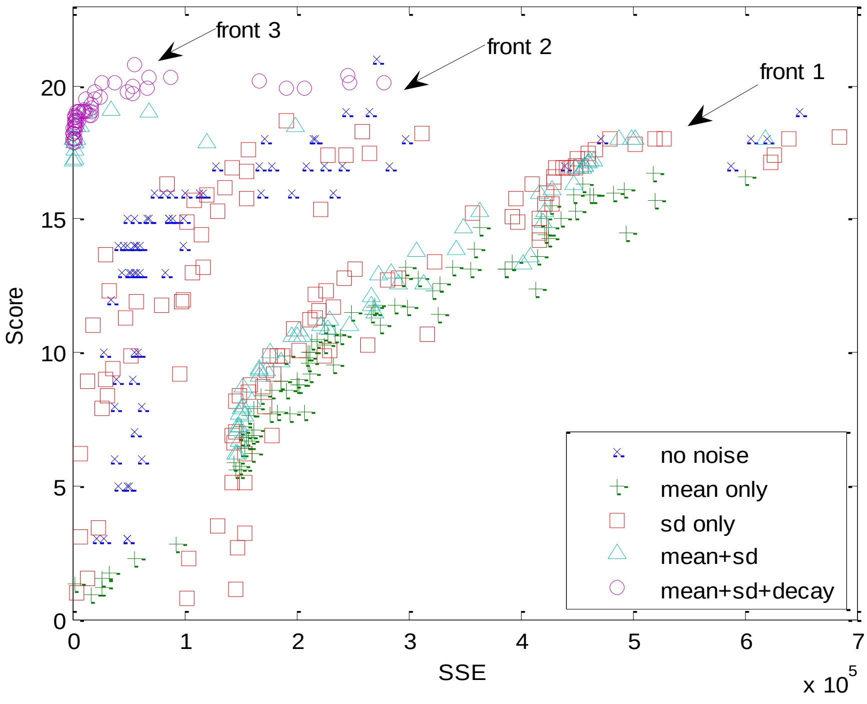
From Figure 6.25, it was observed that H2L matched well with Hu in Histo2 since the SSE of Histo2 was used as the fitness function in this experiment. Although Histo1 was not used as a fitness function, H2L also managed to obtain Histo1 results that were similar to Hu. This was because the action sequences in Histo2 were derived from Histo1. Therefore, a controller that was evolved to match the action sequences in Histo2 will also match the actions in Histo1 as a side effect. The computation time required to evaluate the fitness function was also increased.

6.6.2 Effects of noise

In this section, the effects of introducing sensor noise to the controller will be investigated. The Gaussian sensor noise introduced to the controller could be divided into three components, the mean, the standard deviation, and the decay. Since the noise was introduced to the sensors to imitate errors in human judgment, it will be analyzed in terms of measurement errors in observation.

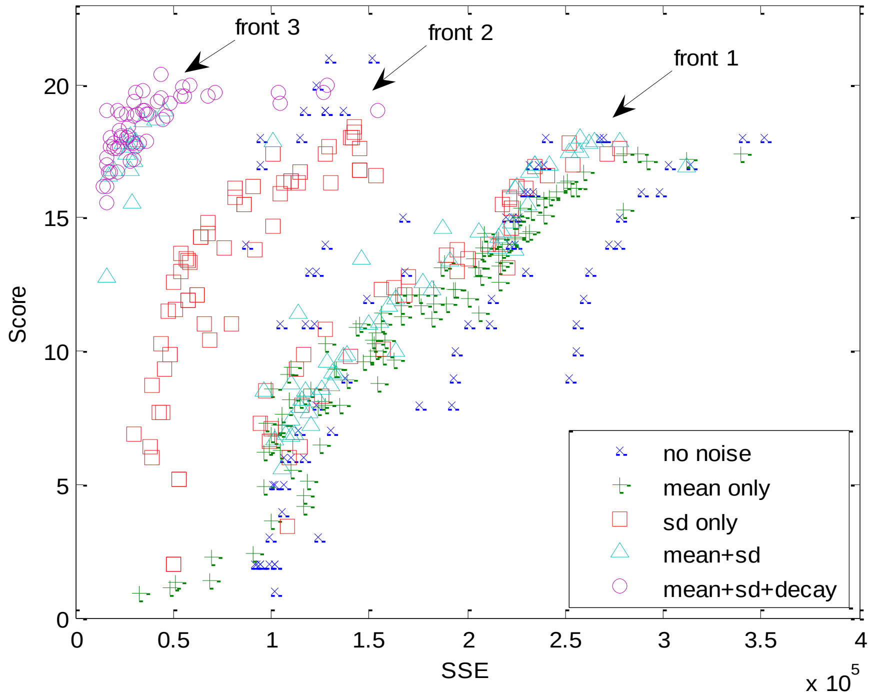
Measurement errors are divided into two components, systematic error and random error. A basic type of systematic error is caused by the incorrect calibration of the measuring instrument. This error is constant and is always present even in separate measurements. The mean of the Gaussian noise introduced to the controller is analogous to systematic errors. On the other hand, random errors are inconsistent in repeated measures and tend to be scattered about the true value. Random errors can be caused by imprecise instruments or subjective interpretation of the instrument reading by the user such as parallax errors. This is analogous to the standard deviation of the Gaussian noise introduced to the controller. Finally, the error in judging distances in a virtual environment increases with distance [20]. This is represented by making the noise a function of the distance to the object. That is, more noise is introduced when observing a distant object and less noise for a nearby object.

It may be interesting to analyze the controller parameters and the noise parameters separately to see how much 'noise' is optimal. However, this was not possible in practice. This was because while the controller and the noise components were designed as separate entities, the evolutionary algorithm saw the problem as a whole. In addition, evolutionary optimization algorithms were known to exploit the dynamics in the inputs and the fitness functions to produce good and unexpected results. Due to this limitation, it was not possible to isolate the noise component from the controller. Consequently, simply looking at the final evolved values of the mean and standard deviations were not meaningful either. As such, each noise component was introduced modularly and its effects analyzed using a black box approach.



Results are plotted at the end of 200 generations for various combinations of sensor noise.

Figure 6.26 Pareto diagram of solutions evolved using waypoints and Histo1 as objectives



Results are plotted at the end of 200 generations for various combinations of sensor noise.

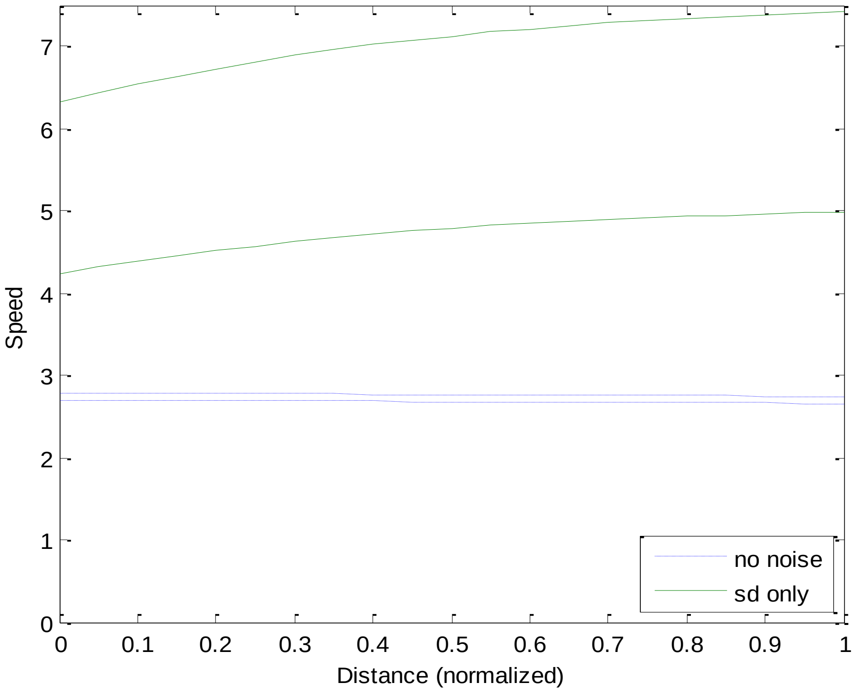
Figure 6.27 Pareto diagram of solutions evolved using waypoints and Histo2 as objectives

The three components of the noise were introduced to the controller in a modular way and the results of the training were summarized in the form of a Pareto diagram. The results for using Histo1 and Histo2 as the fitness function are shown in Figure 6.26 and Figure 6.27 respectively. Each figure has three Pareto fronts labeled front 1, front 2, and front 3 respectively. Front 3 dominants front 2 which dominants front 1.

The effect of evolving only the mean component of the Gaussian noise (systematic error) was first considered. For Histo1, it was observed that evolving mean noise degraded the performance of the controller. That is, the solutions with mean noise (front 1) were completely dominated by the solutions without noise at all (front 2). The result was similar in the case of Histo2 except that the solutions without noise was more scattered (front 1 and 2) and did not consistently converge to an obvious Pareto front. These results suggested that adding only mean noise or simply a constant bias to the sensors did not improve the performance of the controller. This can be appreciated intuitively as using an improperly calibrated instrument in an experiment will do more harm than good.

Next, the effect of evolving the standard deviation component of the Gaussian noise (random error) was considered. In Histo1, evolving only standard deviation resulted in the formation of two Pareto fronts, one of which (front 2) is the same front as that of no noise added and one other (front 1) was the same as that with evolved mean noise. This implied that evolving the standard deviations increased the dimension of the search space and made the fitness landscape more complex such that some solutions became trapped in local optima. However, during good runs, the solutions with evolved standard deviation were as good as those without noise at all. Next, the case of evolving only mean against evolving both mean and standard deviation will be considered. It was observed in Figure 6.26 that evolving both mean and standard deviation produced two Pareto fronts, one (front 1) identical to that evolved with mean only, and a dominant one (front 3) better than that without noise (front 2). This suggested that evolving both mean and standard deviation produced better solutions than the case without noise. This result was also observed in the case of using Histo2 as the fitness function in Figure 6.27.

To better appreciate the reasons for this improvement, the decision regions of the speed regulating component of the controller is plotted in Figure 6.28. For each controller, two lines were plotted. If the instantaneous speed was above both lines, then a brake action was asserted. If the speed was below both lines, an accelerate action was asserted. If the speed was between the lines, a neutral action was asserted. It was observed that for the no noise controller, the two lines were very close together. This meant that either the accelerate action or brake action would be asserted a large percentage of the time while the neutral action would rarely be asserted. For the controller with evolved standard deviation, the two lines were further apart and hence the neutral action was asserted more frequently. Recall that Hu used the neutral action more often than any other actions. Therefore, a controller that asserts the neutral action more frequently (i.e. controller with evolved standard deviation) would obtain a lower SSE and hence was a fitter solution.



If the instantaneous speed is above both lines, then accelerate; below both lines, then brake; otherwise, neutral.

Figure 6.28 Evolved decision space of hyperbolic tangent driving function for the case of no noise and standard deviation only

To further appreciate how such a solution was evolved, the decision space of the controller was examined. Suppose the two decision lines were close together and random noise (i.e. standard deviation) was present. Then for multiple similar situations, the observed sensor measurements would be different every time, both above and below the decision lines. This would result in different actions asserted for the similar situations and hence inconsistent fitness values. As a result, the decision lines were evolved to have a large margin to minimize the difference in asserted actions for similar situations and hence more consistent results. As a side effect, the larger neutral region in the decision space produced lower SSE which further improved the fitness of the solution. This led to the better solutions (front 3) observed in Figure 6.26 and Figure 6.27.

Next, the effect of adding the decay function to the noise was considered. It was observed in both Figure 6.26 and Figure 6.27 that adding the decay function resulted in slightly better solution fronts along front 3 compared to the case without decay (i.e. evolved mean and standard deviation only). It could be said that the best solutions with decay dominates the best solutions without decay in both Histo1 and Histo2. Furthermore, the results for the case with the decay function were more consistent with nearly all its solutions in front 3 compared to the case without decay where only a handful of solutions were located in front 3. In the case of evolved mean and standard deviation without any decay mechanisms, there were soft constraints on both parameters. That is, if the mean (systematic error) or standard deviation (random error) were always present and too large in magnitude, the controller would be unable to score any points because it would not be able to arrive at its desired destination (i.e. the waypoint). The introduction of the decay function ensured that the effects of sensor errors be reduced when the destination was near so that the controller would eventually reach its destination to score a point. In effect, this removed the constraints on the noise parameters and made the fitness landscape less complex, resulting in the better rate of convergence and better solutions.

6.6.3 Generalization

It is important that the evolved controllers not only perform well in training, but they must also be able to perform well in new and unseen situations. As such, it is important to consider the generalization capability of the evolved controllers. Human driving data was collected for 5 different tracks. Only track 1 was used in the training process, the other tracks 2, 3, 4 and 5 were reserved to test the generalization capability of the evolved controllers.

Six controllers were evaluated on all 5 tracks. The six chosen controllers were Hu, H1L, H1H, H2L, H2H, and EH. The comparative results are presented in Table 6.5. The results with the lowest difference in score compared to the human, the lowest mean and standard deviation of SSE of Histo1, and the lowest mean and standard deviation of SSE of Histo2 are highlighted in bold.

It was observed from Table 6.5 that some controllers had waypoint scores of 1 or 0. This meant that these controllers were able to pass through only 1 waypoint or no waypoint at all on the test tracks. This implied that the driving behaviours learnt in the training were not able to generalize well to previously unseen tracks. In this respect, the controllers EH and H2H failed on track 3.

The controllers of more interest to us were the ones with high believability (i.e. low SSE). It was observed that the controllers H1L and H2L were able to obtain waypoint scores that were similar to the human in all 4 test tracks. These controllers were also able to maintain low mean values of SSE in both Histo1 (magnitude of 10^4 or less) and Histo2 (magnitude of 10^5 or less) on all 4 test tracks. Hence, it can be said that both controllers H1L and H2L were able to generalize well to previous unseen tracks. That is, both controllers were able to transfer the knowledge learnt on the training track onto previously unseen tracks.

Comparing only H1L and H2L on each test track, H1L obtained a lower SSE in both Histo1 and Histo2 on test track 2 while H2L obtained a lower SSE in both Histo1 and Histo2 on test tracks 3, 4, and 5. It was also observed that H2L achieved lower standard deviations on all 4 test tracks. That is, on test tracks 3, 4, and 5, H2L which was evolved using Histo2 as its fitness function, obtained a lower SSE in Histo1 compared to H1L despite the fact that H1L was evolved using Histo1 as its fitness function. This implied that controllers evolved using Histo2 as the fitness function produced more robust and consistent controllers. This was because Histo2 was derived from Histo1. Hence, a controller optimized on Histo2 will inadvertently be optimized on Histo1 as well. The reverse was not true. From another perspective, Histo2 contained more information about the human than Histo1. This effect was evident on test tracks 3, 4, and 5 where H2L, which was evolved using Histo2, obtained lower SSE in Histo1 compared to H1L, despite the latter being evolved using Histo1 directly. However, the disadvantage was that the search space was more complicated and the computation time required was longer.

Table 6.5 Comparative results of human driving data, multi-objective controllers, and single objective controllers on training track 1 and testing tracks 2, 3, 4 and 5

All controllers have sensor noise. For waypoint score, the controller with the smallest score difference when compared to the human driving data is highlighted in bold. For sum of square errors (SSE), the controller with the small mean value and the controller with the smallest standard deviation are highlighted in bold. Zero mean scores really mean the controllers scored zero points as they were stucked in an orbit around the first waypoint.

Controller	Score		SSE of Histo1		SSE of Histo2	
	Mean	Std	Mean	Std	Mean	Std
Track 1 (Training data)						
Hu	17.80	0.45	-	-	-	-
H1L	**18.00**	0.47	**3923.76**	1772.01	29208.36	3140.55
H2L	17.40	0.52	5819.96	**1664.82**	**28026.60**	**1688.36**
H1H	20.60	0.70	103420.96	6970.23	95822.40	4884.56
H2H	20.70	0.48	204439.12	1228.00	129596.06	2079.94
EH	21.00	0.00	120746.36	4401.44	109730.56	2239.79
Track 2						
Hu	15.00	0.00	-	-	-	-
H1L	**14.20**	0.42	**7534.56**	2586.61	**40353.76**	2922.87
H2L	14.00	0.00	16034.44	**1626.80**	45882.48	**2785.06**
H1H	18.00	0.00	123563.40	11366.20	128920.44	7515.93
H2H	19.00	0.00	240383.52	3440.06	149929.00	1745.29
EH	16.70	0.95	93029.92	6501.68	113711.60	6256.08
Track 3						
Hu	15.80	0.45	-	-	-	-
H1L	**15.60**	0.84	19205.80	12603.20	45645.08	10211.57
H2L	**16.00**	0.00	**5053.76**	**3809.68**	**33820.44**	**3690.19**
H1H	14.40	0.82	151808.56	31062.31	132936.44	19992.38
H2H	1.00	0.00	819612.80	999.75	782722.80	573.66
EH	0.00	0.00	942713.60	0.00	884091.92	0.00
Track 4						
Hu	16.00	0.00	-	-	-	-
H1L	15.40	0.70	32426.36	16830.25	76525.64	13164.82
H2L	16.60	0.52	**1532.40**	**513.21**	**45084.64**	**2480.23**
H1H	**16.30**	0.48	237105.92	17236.08	240092.20	18210.00
H2H	17.60	0.97	402390.16	10066.62	292699.68	7587.26
EH	13.60	2.55	293268.96	68006.90	292117.40	53485.97
Track 5						
Hu	17.80	0.84	-	-	-	-
H1L	16.70	0.67	7886.04	4766.16	41251.52	5917.33
H2L	16.20	0.42	**5807.20**	**1596.58**	**38763.72**	**3211.29**
H1H	**17.90**	0.57	113853.44	17007.15	114904.20	7083.31
H2H	18.00	0.67	209914.72	8020.79	140758.96	3502.97
EH	18.90	0.32	91323.16	10562.67	108883.88	3684.35

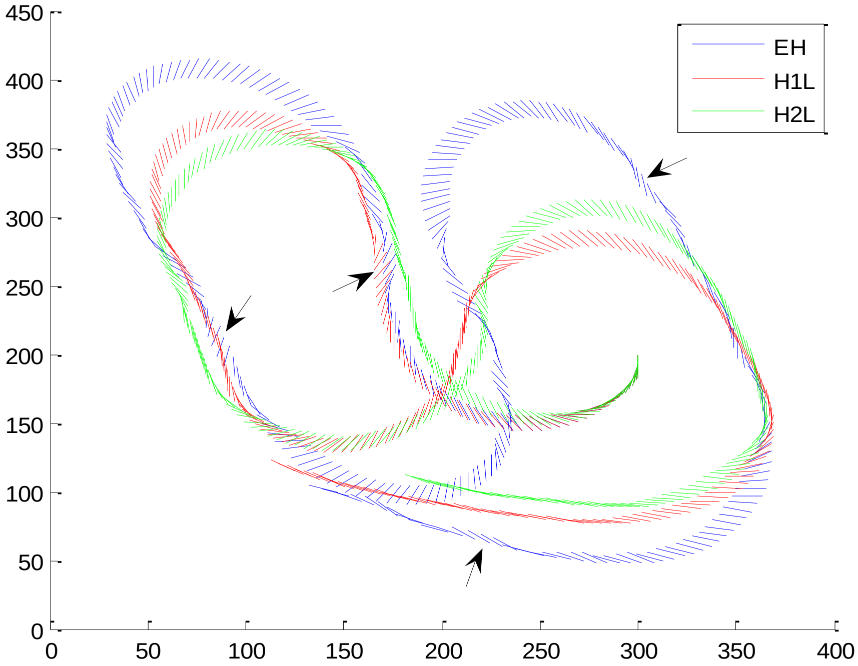


Figure 6.29 Sample trajectories and headings of controllers EH, H1L, and H2L in the first 300 time steps on track 1

It was also useful to subjectively discuss the believability of the evolved controllers by visual observation. The two controllers H1L and H2L were visually observed and compared to controller EH on all 4 test tracks. Sample trajectories and headings of the controllers EH, H1L and H2L in the first 300 time steps are plotted in Figure 6.29 for illustration.

Both controllers learnt to drive in the forward direction unlike the controller ENN that drove in reverse. Both controllers were also able to drive smoothly around the track without the oscillatory behaviour that was observed with the EH and ENN. The oscillatory trajectory of EH can be seen at the regions pointed by arrows in Figure 6.29; the trajectories of H1L and H2L could be observed as smoother in the same regions. The smoother driving behaviour of H1L and H2L could be attributed to the larger distance between the driving and steering decision lines similar to those seen in Figure 6.28. Controllers H1L and H2L were also observed to speed up when they were far away from the waypoint and slow down when approaching the waypoint. Often, when the controllers were near the waypoint, they would output action 5 (neutral) and glided towards the waypoint with its residual momentum. This sometimes gave the impression that the controller was thinking or considering its next move. However, some unnatural behaviour still remained. For instance, the controllers tend to drive in a near perfect circular arc when making turns which seemed too precise to be humanlike. This circular trajectory could also be observed in Figure 6.29. For a more objective measure, a user study conducted to quantify the believability of the evolved controllers will be presented in the next section.

6.6.4 User study

A user study was conducted to objectively quantify the believability of the evolved controllers. Two research questions will be investigated in this study. First, whether the evolved controllers proposed in this chapter are distinguishable as more believable compared to one evolved for performance alone. Second, whether Histo1 or Histo2 as the fitness function evolve the more believable controller.

An objective evaluation of believability is necessary for this study. Riedl & Young proposed an evaluation procedure for multi-agent story generation systems [128] but it is not suitable for evaluating the movement behaviours of game agents. Instead, the believability index proposed in [57] will be used as an objective measure of believability in this study. The detailed procedure and discussion can be found in [57]. Users were asked to watch recorded videos of four controllers. They are the controllers EH, Hu, H1L, and H2L. Each user was first given some time to play the game to familiarize themselves with the workings of the game. Next, the user was asked to estimate their experience with this type of driving game based on the ratings given in Table 6.6. The experience level of the person will be used to weigh his ratings in the overall believability index as well as to calculate the confidence index of the entire study. Next, the user was showed two videos simultaneously, each 51.5 seconds long equivalent to 1000 simulation time steps. The user was specifically instructed that the videos may depict any combination of human and artificial players. The user was then asked to give a rating as shown in Table 6.7 for each video. Each user was shown three pairs of videos of non repeated combinations. A total of 58 people participated in the study and a total of 348 video ratings were collected, of which 1 person's results (i.e. 6 video ratings) was discarded because it was discovered that he misunderstood the instructions. The results are presented in Table 6.8.

Table 6.6 Description of experience level rating of the respondents in the user study

Rating	Description
1	Never play
2	Some passing familiarity
3	Played once monthly
4	Played once weekly
5	Played three times weekly

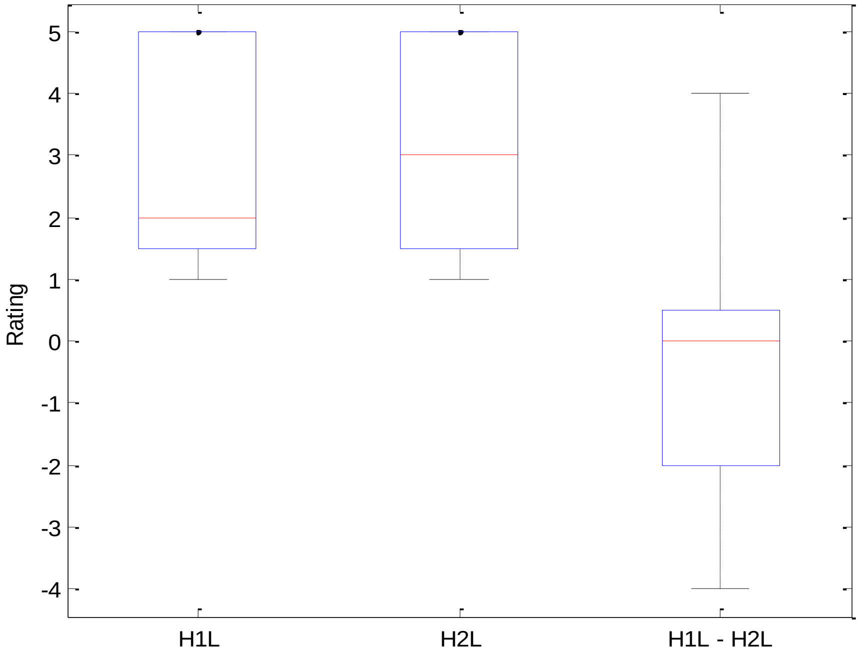
Table 6.7 Description of human-ness rating of the controllers in the user study

Rating	Description
1	Human
2	Probably human
3	Don't know
4	Probably computer
5	Computer

Table 6.8 Believability index of controllers in the user study

Believability index ranges from 0 (least believable) to 1 (most believable). Confidence level of the user study is calculated based on the experience level of the respondents.

Controller	Believability index	Confidence level
EH	0.5178	0.8034
Hu	0.8906	
H1L	0.6311	
H2L	0.5833	



The difference in each pair of ratings was also plotted. A negative rating implied that H2L was more believable than H1L.

Figure 6.30 Boxplot of ratings where H1L and H2L were shown as pairs

It was observed in Table 6.8 that the believability index for controllers EH, Hu, H1L and H2L were 0.5178, 0.8906, 0.6311, and 0.5833 respectively. That is, the controller Hu was correctly identified as human 89.06% of the time while the controller EH was misidentified as human 51.78% of the time. This showed that the users were able to discern between the human and the artificial controller EH. The controllers H1L and H2L were misidentified as human 63.11% and 58.33% of the time respectively. These results were an improvement over the controller EH, implying that the users perceived H1L and H2L as more believable compared to EH. This result provided evidence that the proposed method of using histograms and sensor noise to learn the behavioural tendencies of humans was feasible and that it improved the believability of the controller. However, the results were still some distance from that of an actual human of 89.06%.

Next, the two fitness functions, Histo1 or Histo2, were compared to find out which one evolved the more believable controller. To do this, the results of the user study where controllers H1L and H2L were shown as pairs to the user were examined. There were 28 instances of such a pairing. The distributions of the ratings are presented as a boxplot in Figure 6.30. For conciseness, the difference in each pair of ratings was also plotted in the same diagram. A one tailed paired t-test was performed on the results at the 5% significance level. The p-value did not give sufficient evidence to reject the null hypotheses at this level of significance. Hence, the user study did not reject that the controllers evolved using Histo1 and Histo2 were indifferent. The median and mode of the difference in ratings were both zero, also suggesting that the two controllers were indifferent. Still, some insights on the differences between these two controllers could be inferred from the comments given by the users. Several users commented that they rated Histo2 as more believable than Histo1 because the former traveled at a slower speed. This suggested that the speed profile of a NPC might have effects on its believability. It would be interesting to investigate how the speed profile could be modeled and applied to evolve more believable behaviours.

6.7 Summary

Two main ideas were examined in this chapter. First, sensor noise was introduced to imitate errors in human judgment. The parameters associated with the sensor noise were evolved together with the car controller. This was demonstrated to be feasible and that the introduction of sensor noise can improve the believability of the controller without degrading the driving performance. Each component of the sensor noise was analyzed and the combination of mean, standard deviation and decay was found to produce the best results. Second, the action histogram and action sequence histogram were quantitatively and qualitatively analyzed and associated to some of the unnatural and unrealistic driving behaviours in controllers evolved for performance alone. Hence, it was proposed that the evolved controller can learn to drive more believably by imitating the human driving histogram to learn low level behavioural tendencies of humans. The multi-objective evolution framework was applied to maximize the waypoint score and to minimize the sum of squared errors of the proposed histograms. The controllers were trained using only 1 track and tested on 4 other previously unseen tracks. The controllers selected based on low SSE from the set of Pareto optimal solutions were able to generalize well on all the testing tracks. A user study involving 58 respondents was conducted to objectively quantify the believability of the evolved controllers. The evolved controllers were evaluated as being more believable compared to the controller evolved for performance alone. The proposed action histograms framework was compatible with games using discrete control schemes. Games in this genre included platform games (i.e. Super Mario) and arcade shooters (i.e. Asteroids).

Chapter Seven

7 Conclusion

Enhancing the player experience is an important aspect of developing computer games. This thesis has explored and conducted successful experiments on two key issues affecting the player experience in computer games, namely adaptability and believability, by applying concepts from computational intelligence. This chapter provides a high level summary of the work documented in this thesis and some open directions for future research.

7.1 Summary of experiments

The primary aim of this thesis was to present an investigation on using the computational intelligence approach to enhance the player experience in computer games. A real time car racing simulator game was used as the test bed in the experiments. The real time nature of the test bed required that the game AI be computationally efficiency in addition to traditional performance competency.

Chapter 4 proposed a framework for designing a computationally efficient game AI suitable for implementation in real time games based on a hybrid evolutionary behaviour-based methodology. Genetic algorithm was employed to complement and automate the process of hand designed components required in the behaviour-based methodology. The resulting AI was compared against the popular paradigm of evolutionary neural network and the former was shown to have better performance as well as being more efficient. Genetic algorithm was shown to have successfully exploited some collaboration between the different behaviour components which might have gone unnoticed if it was designed by hand. By benchmarking against the top 5 controllers from the IEEE CEC 2007 Simulated Car Racing competition, the proposed AI was also demonstrated to have good generalization performance. The proposed AI scored the second highest in benchmark performance but was 482 times faster than the top scoring AI. In the subsequent round robin tournament, the proposed AI was able to demonstrate its better generalization capability and outperformed all the other 5 controllers. The advantages of better computational efficiency and generalization performance made the proposed evolutionary behaviour-based framework a suitable candidate for implementation in real time games and laid the groundwork for investigations into adaptability and believability.

Two adaptive algorithms, built upon the proposed framework, were introduced in chapter 5 to address the issue of adaptability in game AI. The adaptive algorithms drew inspirations from reinforcement learning and evolutionary algorithms to improve player satisfaction by scaling the difficulty of the game AI while the game was being played. The advantage was that adaptation was done during the game session itself and no offline training was required. The proposed algorithms also had the advantage of being easily scalable. Two new parameters, learning rate and mutation rate, were introduced by the proposed algorithms. Both parameters were thoroughly investigated and a general rule of thumb for the selection of these two parameters was put forward. Two indicators were also proposed as a measure of an even game between the two players. An analysis of the respective score distributions showed that both algorithms were robust, consistent, and able to generalize well across different types of opponents. The single chromosome variant was shown to be more computationally efficient while the double chromosome variant was more useful in lessening player frustration. Both proposed adaptive algorithms were shown to automatically learn suitable sets of behaviours to adapt to the competency of different opponents, hence keeping the player engaged by continually providing sufficient challenge during the game.

Chapter 6 presented two ideas to induce believable movement behaviours in game agents. First, evolvable sensor noise was used to imitate systematic errors and random errors made by humans. Second, the action histogram and action sequence histogram were proposed as a means to analyze the differences between the unnatural behaviours observed in performance optimized game AI and the behaviours of human players. Subsequently, the histograms were used as fitness functions to induce believable movement behaviours in the game AI by imitating low level behavioural tendencies of human players. It was also demonstrated that performance and believability were conflicting metrics and a multi-objective evolutionary approach was used to improve the believability of the game AI without degrading its performance. Results also showed that the evolution of sensor noise was necessary to encourage humanlike behaviours. A user study involving 58 respondents was conducted to objectively quantify the believability of the evolved game AI and the results verified that the evolved game AI was seen by human players as being more believable.

7.2 Future works

Although computational intelligence techniques have been successfully applied to enhance some aspects of player experience in games, the series of works presented in this thesis barely scratched the surface of what is potentially left to be addressed.

In the experiments involving adaptability, the capacity to match the competency of an opponent necessitates that the game AI be stronger than its opponent. However, human players are good learners and will likely discover ways of defeating the game AI eventually through repeated plays and accumulated experience with the game. In other words, a non-learning game AI places an upper limit on its own level of competency. It also implies that a human player that has learnt to defeat the game AI at its most difficult setting will not be able to benefit from the adaptive game AI. Therefore, a game AI that is able to continually learn and improve together with the human player is desirable. A game AI framework using evolvable fuzzy logic elements is currently being investigated as a possible candidate for such self learning paradigms.

Some direction for future work was obtained from the comments of the user studies in chapter 6. A number of users noticed that the computer players were too fast to react when a new waypoint appeared and hence correctly identified the human player by noticing the delay in reaction times. Loyall defined that the responsiveness of a believable agent must be within the ranges people were willing to accept as believable [86]. Laird and Duchi also determined decision time as a factor affecting human-likeness [81]. It will be an interesting extension to add time delays as a form of both sensor noise and actuator noise to the evolved controllers. Investigations will include the use of static or dynamic time delays, whether delays lead to more believable controllers, and the thresholds of delays that can be added before the controller becomes unstable or uncontrollable. The parameters that characterize the time delays can also be optimized by evolutionary algorithms.

The scenarios explored so far in this thesis involved strictly two player games. However, there are other genres of games in which the human player is competing against numerous opponents in the game. With a greater number of AI controlled opponents, there is greater potential to create a more immersive game experience. For instance, in a three player car racing simulator game, the two AI opponents can collaborate by accidentally colliding into each other, hence giving the human player a better opportunity to reach the current waypoint first. However, care needs to be taken to ensure that such efforts by the game AI do not appear intentional. Otherwise, it may ruin the sense of achievement experienced by the human player.

Game AI does not necessarily imply a computer controlled opponent that plays against a human player in a competitive environment. That is, game AI need not be adversarial in nature. Game AI can also be used to control game characters that play on the human player’s side or are just neutral NPCs. For example, in the context of real time strategy games, a friendly game AI can be used to control a human player’s individual units on the battle ground. This might mean that individual units can automatically and intelligently take cover when under fire, change formations according to combat situations, and retreat when being outnumbered. This will free the human player from the, sometimes mundane, task of having to micro manage every unit on the battlefield and allow the player to make high level strategic decisions in order to defeat his opponent.

Bibliography

[1] Agapitos, A., Togelius, J. and Lucas, S. M., “Evolving Controllers for Simulated Car Racing using Object Oriented Genetic Programming”, Proceedings of the ninth Annual Conference on Genetic and Evolutionary Computation, pp. 1543-1550, 2007.

[2] Agapitos, A., Togelius, J., Lucas, S. M., Schmidhuber, J. and Konstantinidis, A., “Generating Diverse Opponents with Multiobjective Evolution”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 135-142, 2008.

[3] Aler, R., Valls, J. M., Camacho, D. and Lopez, A., “Programming Robosoccer agents be modeling human behavior”, Expert Systems with Applications, vol. 36, pp. 1850-1859, 2009.

[4] Andrade, G., Ramalho, G., Santana, H. and Corruble, V., “Automatic computer game balancing: a reinforcement learning approach”, Proceedings of the fourth International Conference on Autonomous Agents and Multiagent Systems, pp. 1111-1112, 2005.

[5] Andrade, G., Ramalho, G., Santana, H. and Corruble, V., “Challenge-Sensitive Action Selection: an Application to Game Balancing”, Proceedings of IEEE International Conference on Intelligent Agent Technology, pp. 194-200, 2005.

[6] Angeline, P. J., Saunders, G. M. and Pollack, J. B., “An Evolutionary Algorithm that Constructs Recurrent Neural Networks”, IEEE Transactions on Neural Networks, vol. 5, no. 1, pp. 54-65, 1994.

[7] Avery, P. M., Greenwood, G. W. and Michalewicz, Z., “Coevolving Strategic Intelligence”, Proceedings of IEEE Congress on Evolutionary Computation, pp. 3523-3530, 2008.

[8] Bäck, T., “Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms”, Oxford University Press, 1996.

[9] Bain, M. and Sammut, C., “A Framework for Behavioural Cloning”, Machine Intelligence 15, Oxford University Press, pp. 103-129, 1999.

[10] Bakkes, S. and Spronck, P., “Gathering and Utilising Domain Knowledge in Commercial Computer Games”, Proceedings of the Belgium-Netherlands Conference on Artificial Intelligence, pp. 35-42, 2006.

[11] Bakkes, S., Kerbusch, P., Spronck, P. and van den Herik, J., “Automatically Evaluating the Status of an RTS game”, Proceedings of the Annual Belgian-Dutch Machine Learning Conference, pp. 143-144, 2007.

[12] Bakkes, S., Spronck, P. and van den Herik, J., “Rapid Adaptation of Video Game AI”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp 79-86, 2008.

[13] Baluja, S., “Evolution of an Artificial Neural Network Based Autonomous Land Vehicle Controller”, IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, vol. 26, no. 3, pp. 450-463, 1996.

[14] Barber, H. and Kudenko, D., “Generation of Adaptive Dilemma-Based Interactive Narratives”, IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 4, pp. 309-326, 2009.

[15] Batavia, P. H., Pomerleau, D. A. and Thorpe, C. E., “Applying Advanced Learning Algorithms to ALVINN”, technical report CMU-RI-TR-96-31, Robotics Institute, Carnegie Mellon University, 1996.

[16] Bellotti, R., Berta, R., De Gloria, A. and Primavera, L., “Adaptive Experience Engine for Serious Games”, IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 4, pp. 264-280, 2009.

[17] Bergsma, M. and Spronck, P., “Adaptive Intelligence for Turn-based Strategy Games”, Proceedings of the Belgian-Dutch Artificial Intelligence Conference, pp. 17-24, 2008.

[18] Beume, N., Danielsiek, H., Eichhorn, C., Naujoks, B., Preuss, M., Stiller, K. and Wessing, S., “Measuring Flow as Concept for Detecting Game Fun in the Pac-Man Game”, Proceedings of IEEE Congress on Evolutionary Computation, pp. 3447-3454, 2008.

[19] Bhatt, K., “Believability in Computer Games”, Proceedings of the first Australian Workshop on Interactive Entertainment, pp.81-84, 2004.

[20] Bodenheimer, B., Meng, J., Wu, H., Narasimham, G., Rump, B., McNamara, T. P., Carr, T. H. and Rieser, J. J., “Distance Estimation in Virtual and Real Environments using Bisection”, Proceedings of the Fourth Symposium on Applied Perception in Graphics and Visualization, pp. 35-40, 2007.

[21] Braathen, S. and Sendstad, O. J., “A Hybrid Fuzzy Logic/Constraint Satisfaction Problem Approach to Automatic Decision Making in Simulated Game Models”, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 34, no. 4, pp. 1786-1797, 2004.

[22] Brooks, R. A., “A Robust Layered Control System for a Mobile Robot”, IEEE Journal of Robotics and Automation, vol. 2, no. 1, pp. 14-23, 1986.

[23] Bryant, B. D. and Miikkulainen, R., “Neuroevolution for Adaptive Teams”, Proceedings of IEEE Congress on Evolutionary Computation, vol. 3, pp. 2194-2201, 2003.

[24] Bryant, B. D., “Evolving Visibly Intelligent Behavior for Embedded Game Agents”, Ph.D. thesis, Department of Computer Sciences, University of Texas, Austin, TX, 2006.

[25] Bryant, B. D. and Miikkulainen, R., “Acquiring Visibly Intelligent Behavior with Example-Guided Neuroevolution”, Proceedings of the Twenty-Second National Conference on Artificial Intelligence, pp. 801-808, 2007.

[26] Bryson, J. J, “The Behavior-Oriented Design of Modular Agent Intelligence”, Agent Technologies, Infrastructures, Tools, and Applications for E-Services, pp. 61-76, 2002.

[27] Buro, M. and Furtak, T., “RTS Games as Test-Bed for Real-Time AI Research”, Proceedings of the Seventh Joint Conference on Information Science, pp. 481-484, 2003.

[28] Buro, M., “Call for AI Research in RTS Games”, Proceedings of the Association for the Advancement of Artificial Intelligence Workshop on AI in Games, pp. 139-142, 2004.

[29] Cardamone, L., Loracono, D. and Lanzi, P. L., “Learning Drivers for TORCS through Imitation Using Supervised Methods”, IEEE Symposium on Computational Intelligence and Games, pp. 148-155, 2009.

[30] Chaperot, B. and Fyfe, C., “Advanced artificial intelligence techniques applied to a motocross game”, Computing and Information Systems, vol. 10, no. 2, pp. 27-31, 2006.

[31] Charles, D., McNeill, M., McAlister, M., Black, M., Moore, A., Stringer, K., Kücklich, J. and Kerr, A., “Player-Centred Game Design: Player Modelling and Adaptive Digital Games”, Digital Games Research Conference, pp. 285-298, 2005.

[32] Chellapilla, K. and Fogel, D. B., “Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge”, IEEE Transactions on Neural Networks, vol. 10, no. 6, pp. 1382-1391, 1999.

[33] Chellapilla, K. and Fogel, D. B., “Evolving an expert checkers playing program without using human expertise”, IEEE Transaction of Evolutionary Computation, vol. 4, pp. 422-428, 2001.

[34] Choi, D., Konik, T., Nejati, N., Park, C. and Langley, P., “A Believable Agent for First-Person Shooter Games”, Proceedings of the third Artificial Intelligence and Interactive Digital Entertainment International Conference, pp. 71-73, 2007.

[35] Chong, S. Y., Tiño, P. and Yao, X., “Measuring Generalization Performance in Co-evolutionary Learning”, IEEE Transactions on Evolutionary Computation, vol. 12, no. 4, pp. 479-505, 2008.

[36] Coello Coello, C. A., "A Short Tutorial on Evolutionary Multiobjective Optimization", Proceedings of the first International Conference on Evolutionary Multi-Criterion Optimization, pp. 21-40, 2001.

[37] Cole, N., Louis, S. J. and Miles, C., "Using a genetic algorithm to tune first-person shooter bots", Proceedings of IEEE Congress on Evolutionary Computation, pp. 139-145, 2004.

[38] Csikszentmihályi, M., "Flow: The Psychology of Optimal Experience", New York: HaperCollins, 1990.

[39] De Jong, K. A., "An Analysis of the Behavior of a Class of Genetic Adaptive Systems", Ph.D. thesis, University of Michigan, Ann Arbor, Mich, 1975.

[40] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T., "A Fast and Elitist Multiobjective Genetic Algorithm: NSGAII," IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, 2002.

[41] Dezinger, J. and Kordt, M., "Evolutionary On-line Learning of Cooperative Behavior with Situation-Action-Pairs", Proceedings of the fourth International Conference on MultiAgent Systems, pp. 103-110, 2000.

[42] DeSouza, G. N. and Kak, A. C., "A Subsumptive, Hierarchical, and Distributed Vision-Based Architecture for Smart Robotics", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 34, no. 5, pp. 1988-2002, 2004.

[43] Duro, J. A. and de Oliveira, J. V., "Particle Swarm Optimization Applied to the Chess Games", Proceedings of IEEE Congress on Evolutionary Computation, pp. 3702-3709, 2008.

[44] Entertainment Software Association, Industry Facts, Economic Data, http://www.theesa.com/facts/econdata.asp, retrieved on 26 May 2010.

[45] Fernández, A. J. and González, J. J., "Action Games: Evolutive Experiences", Computational Intelligence, Theory and Applications, vol. 33, pp. 487-501, 2005.

[46] Fernlund, H. K. G., Gonzalez, A. J., Georgiopoulos, M. and DeMara, R. F., "Learning Tactical Human Behavior Through Observation of Human Performance", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 36, no. 1, pp. 128-140, 2006.

[47] Fogel, D. B., Hays, T. J., and Johnson, D. R., "A platform for evolving intelligently interactive adversaries", Biosystems, vol. 85, no. 1, pp. 72-83, 2006.

[48] Fonseca, C. M. and Fleming, P. J., "Genetic algorithm for multiobjective optimization, formulation, discussion and generalization", Proceedings

of the fifth International Conference on Genetic Algorithms, pp. 416-423, 1993.

[49] Forbus, K. and Laird, J., "AI and the entertainment industry", IEEE Intelligent Systems, vol. 17, no. 4, pp. 15-16, 2002.

[50] Fujii, S., Nakashima, T. and Ishibuchi, H., "A Study on Constructing Fuzzy Systems for High-Level Decision Making in a Car Racing Game", Proceedings of IEEE Congress on Evolutionary Computation, pp. 3626-3633, 2008.

[51] Ghoneim, A., Abbass, H. and Barlow, M., "Characterizing Game Dynamics in Two-Player Strategy Games Using Network Motifs", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 38, no. 3, pp 682-690, 2008.

[52] Goh, C. K., "Evolutionary multi-objective optimization in uncertain environments", Ph.D. thesis, Department of Electrical & Computer Engineering, National University of Singapore, 2007.

[53] Goh, C. K., Tan, K. C. and Tay, A., "A Competitive-Cooperation Coevolutionary Paradigm for Multi-objective Optimization", Proceedings of IEEE International Symposium on Intelligent Control, pp. 255-260, 2007.

[54] Goh, C. K., Teoh, E. J. and Tan, K. C., "Hybrid multiobjective evolutionary design for artificial neural networks", IEEE Transactions on Neural Networks, vol. 19, no. 9, pp. 1531-1548, 2008.

[55] Gomez, F., Schmidhuber, J. and Miikkulainen, R., "Efficient Non-Linear Control through Neuroevolution", Proceedings of the European Conference on Machine Learning, pp. 654-662, 2006.

[56] Gomez, F. J., Togelius, J. and Schmidhuber, J., "Measuring and Optimizing Behavioural Complexity for Evolutionary Reinforcement Learning", Proceedings of the International Conference on Artificial Neural Networks, pp. 765-774, 2009.

[57] Gorman, B., Thurau, C., Bauckhage, C. and Humphrys, M., "Believability Testing and Bayesian Imitation in Interactive Computer Games", From Animals to Animats 9, vol. 4095, pp. 655-666, 2006.

[58] Guesgen, H. W. and Shi, X. D., "An Artificial Neural Network for a Tank Targeting System", Proceedings of the International FLAIRS Conference, pp. 463-464, 2006.

[59] Gwiazda, T. D., "Genetic Algorithms Reference Vol. 1 Crossover for single-objective numerical optimization problems", Tomasz Gwiazda, Lomianki, 2006.

[60] Hagelbäck, J. and Johansson, S. J., "Measuring player experience on runtime dynamic difficulty scaling in an RTS game", Proceedings of the

fifth International Conference on Computational Intelligence and Games, pp. 46-52, 2009.

[61] Hastings, E. J., Guha, R. K. and Stanley, K. O., "Evolving content in the galactic arms race video game", Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 241-248, 2009.

[62] Hastings, E. J., Guha, R. K. and Stanley, K. O., "Automatic Content Generation in the Galactic Arms Race Video Game", IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 4, pp. 245-263, 2009.

[63] Haykins, S., Neural Networks: A Comprehensive Foundation New York: MacMillan, 1994.

[64] Hillis, W. D., "Co-evolving parasites improve simulated evolution as an optimization procedure", Proceedings of the ninth annual International Conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks on Emergent Computation, pp. 228-234, 1990.

[65] Ho, D. T. and Garibaldi, J. M., "A Fuzzy Approach for the 2007 CIG Simulated Car Racing Competition", Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 127-134, 2008.

[66] Ho, D. T. and Garibaldi, J. M., "A Novel Fuzzy Inferencing Methodology for Simulated Car Racing", Proceedings of IEEE International Conference on Fuzzy Systems, pp. 1909-1916, 2008.

[67] Holland, J. H., "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence", MIT Press, 1992.

[68] Hong, T. P., Huang, K. Y. and Lin, W. Y., "A Genetic Search Method for Multi-Player Game Playing", Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, vol. 5, pp. 3858-3861, 2000.

[69] Horswill, I. D. and Zubek, R., "Robot Architectures for Believable Game Agents", Proceedings of AAAI Spring Symposium on Artificial Intelligence and Computer Games, Technical Report SS-99-02, 1999.

[70] Hoshino, Y. and Kamei, K., "A proposal of reinforcement learning system to use knowledge effectively", Proceedings of SICE Annual Conference, vol. 2, pp. 1582-1585, 2003.

[71] Huang, B. Q., Cao, G. Y. and Guo, M., "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance", Proceedings of the fourth International Conference on Machine Learning and Cybernetics, pp. 85-89, 2005.

[72] Hughes, E. J., “Checkers using a Co-evolutionary On-Line Evolutionary Algorithm”, Proceedings of IEEE Congress on Evolutionary Computation, pp. 1899-1905, 2005.

[73] Hunicke, R. and Chapman, V., “AI for Dynamic Difficulty Adjustment in Games”, Challenges in Game Artificial Intelligence AAAI Workshop, pp. 91-96, 2004.

[74] IBM Research, Deep Blue, http://www.research.ibm.com/deepblue/, retrieved on 15 May 2007.

[75] Ishibuchi, H., Nakashima, T. and Kuroda, T., “A Hybrid Fuzzy Genetics-based Machine Learning Algorithm: Hybridization of Michigan Approach and Pittsburg Approach”, Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, vol. 1, pp. 296-301, 1999.

[76] Isla, D. and Blumberg, B., “New Challenges for Character-Based AI for Games”, Proceedings of AAAI Spring Symposium on AI and Interactive Entertainment, pp. 41-45, 2002.

[77] Juang, C. F. and Lu, C. F., “Fuzzy Controller Design by Hybrid Evolutionary Learning Algorithms”, Proceedings of IEEE International Conference on Fuzzy Systems, pp. 525-529, 2005.

[78] Knowles, J. D. and Corne, D. W., “Approximating the non-dominated front using the Pareto archived evolution strategy,” Evolutionary Computation, vol. 8, no. 2, pp. 149-172, 2000.

[79] Koster, R., “A theory of fun for game design”, Paraglyph press, 2005.

[80] Laird, J. E. and van Lent, M., “Human-Level AI’s Killer Application: Interactive Computer Games”, Proceedings of the seventeenth Nation Conference on Artificial Intelligence and twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 1171-1178, 2000.

[81] Laird, J. E. and Duchi J. C., “Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot”, in AAAI Fall Symposium Series on Simulating Human Agents, pp. 54-58, 2000.

[82] Langley, P., Laird, J. E. and Rogers, S., “Cognitive architectures: Research issues and challenges”, Cognitive Systems Research, vol. 10, no. 2, pp. 141-160, 2009.

[83] Li, S. T. and Chen, S. C., “Function Approximation using Robust Wavelet Neural Networks”, Proceedings of IEEE International Conference on Tools with Artificial Intelligence, pp. 483-488, 2002.

[84] Lidén, L., “Artificial Stupidity: The Art of Intentional Mistakes”, AI Game Programming Wisdom 2, Charles River Media, 2004.

[85] Livingstone, D., “Turing’s test and believable AI in games”, Computers in Entertainment, vol. 4, no. 1, 2006.

[86] Loyall, A. B., “Believable Agents: Building Interactive Personalities”, Ph.D. thesis, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburg, 1997.

[87] Lubberts, A. and Miikkulainen, R., “Co-Evolving a Go-Playing Neural Network”, Genetic and Evolutionary Computation Conference Workshop, pp. 14-19, 2001.

[88] Lucas, S. M., “Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 203-210, 2005.

[89] Lucas, S. M. and Kendall, G., “Evolutionary Computation and Games”, IEEE Computational Intelligence Magazine, pp. 10-18, 2006.

[90] Magoulas, G. D., Plagianakos, V. P. and Vrahatis, M. N., “Hybrid Methods Using Evolutionary Algorithms for On-line Training”, Proceedings of IEEE International Conference on Neural Networks, pp. 2218-2223, 2001.

[91] Malone, T. W., “What makes things fun to learn? Heuristics for designing instructional computer games”, Proceedings of the third ACM SIGSMALL Symposium and the 1st SIGPC Symposium on Small Systems, pp. 162-169, 1980.

[92] Mantere, T. and Koljonen, J., “Solving and analyzing Sudokus with cultural algorithms”, Proceedings of IEEE Congress on Evolutionary Computation, pp. 4053-4060, 2008.

[93] Mateas, M., “An Oz-Centric Review of Interactive Drama and Believable Agents”, Technical Report CMU-CS-97-156, School of Computer Science, Carnegie Mellon University, Pittsburg, United States, 1997.

[94] Miikkulainen, R., Bryant, B. D., Cornelius, R., Karpov, I. V., Stanley, K. O. and Yong, C. H., “Computational Intelligence in Games”, Computational Intelligence: Principles and Practice, IEEE Computational Intelligence Society, pp. 155-191, 2006.

[95] Miles, J. D. and Tashakkori, R., “Improving the Believability of Non-Player Characters in Simulations”, Proceedings of the second Conference on Artificial General Intelligence, pp. 1-2, 2009.

[96] Miller, B. L. and Goldberg, D. E., “Genetic Algorithms, Selection Schemes, and the varying Effects of Noise”, Evolutionary Computation, vol. 4, no. 2, pp. 113-131, 1996.

[97] Moraglio, A. and Togelius, J., “Geometric Particle Swarm Optimization for the Sudoku Puzzle”, Proceedings of the Annual Conference on Genetic and Evolutionary Computation, pp. 118-125, 2007.

[98] Muñoz, J., Gutierrez, G. and Sanchis, A., “Controller for TORCS created by imitation”, IEEE Symposium on Computational Intelligence and Games, pp. 271-278, 2009.

[99] Murray, J. H., “Hamlet on the Holodeck”, The Free Press, New York, United States, 1997.

[100] Musliner, D. J., Hendler, J. A., Agrawala, A. K., Durfee, E. H., Strosnider, J. K. and Paul. C. J., “The Challenges of Real-Time AI”, IEEE Computer, vol. 28, pp. 58-66, 1995.

[101] Nakashima, T., Takatani, M., Udo, M. and Ishibuchi, H., “An evolutionary approach for strategy learning in robocup soccer”, IEEE International Conference on Systems, Man and Cybernetics, vol. 2, pp. 2023-2028, 2004.

[102] Nakashima, T., Udo, M. and Ishibuchi, H., “A fuzzy reinforcement learning for a ball interception problem”, Lecture Notes in Computer Science, RoboCup 2003: Robot Soccer World Cup VII, vol. 3020, pp. 559-567, 2004.

[103] Nakashima, T., Yokota, Y., Shoji, Y. and Ishibuchi, H., “A genetic approach to the design of autonomous agents for futures trading”, Artificial Life and Robotics, vol. 11, no. 2, pp.145-148, 2007.

[104] Nareyek, A., “Game AI is Dead. Long Live Game AI!”, IEEE Intelligent Systems, vol. 22, no. 1, pp. 9-11, 2007.

[105] Nitschke, G., “Co-evolution of cooperation in a Pursuit Evasion Game”, Proceedings of IEEE Conference on Intelligent Robots and Systems, pp. 2037-2042, 2003.

[106] Olesen, J. K., Yannakakis, G. N. and Hallam, J., “Real-time challenge balance in an RTS game using rtNEAT”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 87-94, 2008.

[107] Ong. C. S., Quek, H. Y., Tan, K. C. and Tay, A., “Discovering Chinese Chess strategies through co-evolutionary approaches”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 360-367, 2007.

[108] Parker, G. B. and Parker, M., “Evolving Parameters for Xpilot Combat Agents”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 238-243, 2007.

[109] Pedersen, C., Togelius, J. and Yannakakis, G. N., “Modeling Player Experience in Super Mario Bros”, Proceedings on IEEE Symposium on Computational Intelligence and Games, pp. 132-139, 2009.

[110] Pedersen, C., Togelius, J. and Yannakakis, G. N., "Optimizing of platform game levels for player experience", Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE'09), 2009.

[111] Pedersen, C., Togelius, J. and Yannakakis, G. N., "Modeling Player Experience for Content Creation", IEEE Transactions on Computational Intelligence and AI in Games, vol. 2, no. 1, pp. 54-67, 2010.

[112] Perez, D., Recio, G., Saez, Y. and Isasi, P., "Evolving a Fuzzy Controller for a Car Racing Competition", Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 263-270, 2009.

[113] Plant, W. R., Schaefer, G. and Nakashima, T., "An Overview of Genetic Algorithms in Simulation Soccer", Proceedings of IEEE Congress on Evolutionary Computation, pp. 3897-3904, 2008.

[114] Ponsen, M., Muñoz-Avila, H., Spronck, P. and Aha, D. W., "Automatically Generating Game Tactics via Evolutionary Learning", AI Magazine, vol. 27, no. 3, pp. 75-84, 2006.

[115] Ponsen, M. and Spronck, P., "Improving Adaptive Game AI with Evolutionary Learning", Proceedings of Computer Games: Artificial Intelligence, Design and Education, pp. 389-396, 2004.

[116] Priesterjahn, S., Kramer, O., Weimer, A. and Goebels, A., "Evolution of Human-Competitive Agents in Modern Computer Games", Proceedings of IEEE Congress of Evolutionary Computation, pp. 777-784, 2006.

[117] Priesterjahn, S. and Eberling, M., "Imitation Learning in Uncertain Environments", Proceedings of the tenth International Conference on Parallel Problem Solving from Nature, pp. 950-960, 2008.

[118] Prieto, C. E., Nino, F. and Quintana, D., "A goalkeeper strategy in robot soccer based on Danger Theory", Proceedings of IEEE Congress on Evolutionary Computation, pp. 3443-3447, 2008.

[119] Quek, H. Y. and Goh, C. K., "Adaptation of Iterated Prisoners Dilemma Strategies by Evolution and Learning", Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 40-47, 2007.

[120] Quek, H. Y., Tan, K. C., and Tay, A., "Public Goods Provision: An Evolutionary Game Theoretic Study under Asymmetric Information", IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 2, pp. 105-120, 2009.

[121] Ramsey, M., "Designing a Multi-Tier AI Framework", AI Game Programming Wisdom 2, Charles River Media, 2003.

[122] Ranganathan, A. and Koenig, S., "A Reactive Robot Architecture with Planning on Demand", Proceedings of IEEE International Conference on Intelligent Robots and Systems, pp. 1462-1468, 2003.

[123] Rani, P., Sarkar, N. and Liu, C., "Maintaining Optimal Challenge in Computer Games Through Real-Time Physiological Feedback", Proceedings of the eleventh International Conference on Human Computer Interaction, pp. 184-192, 2005.

[124] Rechenberg, I., Evolutionsstrategie '94 Stuttgart, Germany: Frommann-Holzboog, 1994.

[125] Ren, J., McIsaac, K. A., Patel, R. V. and Peters, T. M., "A Potential Field Model Using Generalized Sigmoid Functions", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 37, no. 2, pp. 477-484, 2007.

[126] Richards, N., Moriarty, D. E. and Miikkulainen, R., "Evolving Neural Networks to Play Go", Proceedings of the seventh International Conference on Genetic Algorithms, pp. 768-775, 1998.

[127] Riedl, M. O. and Stern, A., "Believable Agents and Intelligent Story Adaptation for Interactive Storytelling", Proceedings of the third International Conference on Technologies for Interactive Digital Storytelling and Entertainment, 2006.

[128] Riedl, M. O. and Young, R. Y., "An objective character believability evaluation procedure for multi-agent story generation systems", Proceedings of the fifth International Working Conference on Intelligence Virtual Agents, pp. 278-291, 2005.

[129] Rizzo, P., Veloso, M., Miceli, M. and Cesta, A., "Goal-Based Personalities and Social Behaviors in Believable Agents", Applied Artificial Intelligence, vol. 13, pp. 239-272, 1999.

[130] Rosin, C. D. and Belew, R. K., "New methods for competitive coevolution", Evolutionary Computation, vol. 5, no. 1, pp. 1-29, 1997.

[131] Runarsson, T. P. and Lucas, S. M., "Coevolution Versus Self-Play Temporal Difference Learning for Acquiring Position Evaluation in Small-Board Go", IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, pp. 628-640, 2005.

[132] Sánchez-Ruiz, A., Lee-Urban, S., Muñoz-Avila, H., Díaz-Agudo, B. and González-Calero, P., "Game AI for a Turn-based Strategy Game with Plan Adaptation and Ontology-based retrieval", Proceedings of the ICAPS Workshop on Planning in Games, 2007.

[133] Sato, Y. and Kanno, R., "Event-driven Hybrid Learning Classifier Systems for Online Soccer Games", Proceedings of IEEE Congress on Evolutionary Computation, vol. 3, pp. 2091-2098, 2005.

[134] Sato, Y., Suzuki, R. and Akatsuka, Y., "Formation Dependency in Event-driven Hybrid Learning Classifier Systems for Soccer Video

Games", Proceedings of IEEE Congress on Evolutionary Computation, pp. 1831-1838, 2008.

[135] Schadd, F., Bakkes, S. and Spronck, P., "Opponent Modeling in Real-Time Strategy Games", Proceedings of the eighth International Conference on Intelligent Games and Simulation, pp. 61-68, 2007.

[136] Schaeffer, J., "One Jump Ahead: Challenging Human Supremacy in Checkers", Springer-Verlag, 1997.

[137] Schaeffer, J., "A gamut of games", Artificial Intelligence Magazine, vol. 22, no. 3, pp. 29-46, 2001.

[138] Scheutz, M. and Andronache, V., "Architecture Mechanisms for Dynamic Changes of Behavior Selection Strategies in Behavior-Based Systems", IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, vol. 34, no. 6, pp. 2377-2395, 2004.

[139] Schrum, J. and Miikkulainen, R., "Evolving multi-model behavior in NPCs", Proceedings of Symposium on Computational Intelligence and Games, pp. 325-332, 2009.

[140] Scott, B., "The Illusion of Intelligence", AI Game Programming Wisdom, Charles River Media, pp. 16-20, 2002.

[141] Sengers, P., "Do the thing right: An architecture for action expression", Proceedings of the Second International Conference on Autonomous Agents, pp. 24-31, 1998.

[142] Sharabi, S. and Sipper, M., "GP-Sumo: Using genetic programming to evolve sumobots", Genetic Programming and Evolvable Machines, vol. 7, no. 3, pp. 211-230, 2006.

[143] Sinclair, M. C., "Evolutionary Algorithms for Optical Network Design: A Genetic-algorithm/heuristic hybrid approach", Ph.D. thesis, University of Essex, 2001.

[144] Spronck, P., Sprinkhuizen-Kuyper, I. and Postma, E., "Online Adaptation of Game Opponent AI in Simulation and in Practice", Proceedings of the fourth International Conference on Intelligence Games and Simulation, pp. 93-100, 2003.

[145] Spronck, P., Sprinkhuizen-Kuyper, I. and Postma, E., "Difficulty scaling of Game AI", 5th International Conference Intelligent Games and Simulation, pp. 33-37, 2004.

[146] Spronck, P., "A Model for Reliable Adaptive Game Intelligence", Proceedings of International Joint Conferences on Artificial Intelligence Workshop on Reasoning, Representation, and Learning in Computer Games, pp. 95-100, 2005.

[147] Spronck, P., “Adaptive Game AI”, Ph.D. thesis, Maastricht University Press, 2005.

[148] Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I. and Postma, E., “Adaptive game AI with dynamic scripting”, Machine Learning, vol. 63, no. 3, pp. 217-248, 2006.

[149] Stanley, K. O. and Miikkulainen, R., “Efficient Reinforcement Learning through Evolving Neural Network Topologies”, Proceedings of the Genetic and Evolutionary Computation Conference, pp. 569-577, 2002.

[150] Stanley, K. O. and Miikkulainen, R., “Evolving neural networks through augmenting topologies”, Evolutionary Computation, vol. 10, no. 2, pp. 99-127, 2002.

[151] Stanley, K. O., “Efficient evolution of neural networks through complexification”, Ph.D. thesis, Department of Computer Sciences, University of Texas, Austin, TX, 2004.

[152] Stanley, K. O., Bryant B. D. and Miikkulainen, R., “Real-time neuroevolution in the NERO video game”, IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, pp. 653-668, 2005.

[153] Stanley, K. O., Bryant, B. D., Karpov I. and Miikkulainen, R., “Real-Time Evolution of Neural Networks in the NERO Video Game”, Proceedings of the Twenty-First National Conference in Artificial Intelligence, pp. 1671-1674, 2006.

[154] Stone, P., Sutton, R. S. and Kuhlmann, G., “Reinforcement learning for RoboCup-soccer keepaway”, Adaptive Behaviour, vol. 13, no. 3, pp. 165-188, 2005.

[155] Sturtevant, N., “A Comparison of Algorithms for Multi-player Games”, Proceedings of the third International Conference on Computers and Games, pp. 108-122, 2003.

[156] Sutton, R. S., McAllester, D., Singh, S. and Mansour, Y., “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Advances in Neural Information Processing Systems, vol. 12, pp. 1057-1063, 2000.

[157] Sweetser, P., Johnson, D., Sweetser, J. and Wiles, J., “Creating engaging artificial characters for games”, Proceedings of the second International Conference on Entertainment Computing, pp. 1-8, 2003.

[158] Sweetser, P. and Johnson, D., “Player-Centered Game Environments: Assessing Player Opinions, Experiences and Issues”, Entertainment Computing, pp. 305-336, 2004.

[159] Sweetser, P. and Wiles, J., “Combining Influence Maps and Cellular Automata for Reactive Game Agents”, Proceedings of Intelligent Data Engineering and Automated Learning, pp. 524-531, 2005.

[160] Szita, I., Ponsen, M. and Spronck, P., "Keeping Adaptive Game AI interesting", Proceedings of CGAMES, pp. 70-74, 2008.

[161] Tan, C. H., Ang, J. H., Tan, K. C. and Tay, A., "Online Adaptive Controller for Simulated Car Racing", Proceedings of IEEE Congress on Evolutionary Computation, pp. 3635-3642, 2008.

[162] Tan, C. H., Ramanathan, K., Guan, S. U. and Bao, C., "Recursive Hybrid Decomposition with Reduced Pattern Training", International Journal of Hybrid Intelligent Systems, vol. 6, no. 3, pp. 135-146, 2009.

[163] Tan, C. H., Tan, K. C. and Tay, A., "Computationally Efficient Behaviour Based Controller for Real Time Car Racing Simulation", Expert Systems with Applications, vol. 37, no. 7, pp. 4850-4859, 2010.

[164] Tan, K. L, Tan, C. H., Tan, K. C. and Tay, A., "Adaptive Game AI for Gomoku", Proceedings of the Fourth International Conference on Autonomous Robots and Agents, pp. 507-512, 2009.

[165] Tan, M., "Multi-agent reinforcement learning: independent vs. cooperative agents", Proceedings of the tenth International Conference on Machine Learning, pp. 330-337, 1997.

[166] Tang, H., Tan, C. H., Tan, K. C. and Tay, A., "Neural Network versus Behavior Based Approach in Simulated Car Racing Game", Proceedings of IEEE Workshop on Evolving and Self-Developing Intelligent Systems, pp. 58-65, 2009.

[167] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J.,Fong, P., Gale, J., Halpenny, M.,Hoffmann, G.,Lau, K., Oakley, C.,Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L. E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A. and Mahoney, P., "Stanley: The Robot that Won the DARPA Grand Challenge", Journal of Field Robotics, vol. 23, no. 9, pp. 661-692, 2006.

[168] Thue, D., Bulitko, V., Spetch, M. and Wasylishen, E., "Interactive Storytelling: A Player Modelling Approach", Proceedings of the Artificial Intelligence and Interactive Digital Entertainment, pp. 43-48, 2007.

[169] Thue, D., Bulitko, V., Spetch, M. and Wasylishen, E., "Learning Player Preferences to Inform Delayed Authoring", Proceedings from AAAI Symposium on Intelligence Narrative Technologies, pp. 158-161, 2007.

[170] Thurau, C., Sagerer, G. and Bauckhage, C., "Imitation learning at all levels of Game-AI", Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education, pp. 402-408, 2004.

[171] Togelius, J. and Lucas, S. M., “Evolving Controllers for Simulated Car Racing”, Proceedings of IEEE Congress on Evolutionary Computation, vol. 2, pp. 1906-1913, 2005.

[172] Togelius, J., De Nardi, R. and Lucas, S. M., “Making racing fun through player modeling and track evolution”, Proceedings of the SAB Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games, 2006.

[173] Togelius, J. and Lucas, S. M., “Arms races and car races”, Lecture Notes in Computer Science, Parallel Problem Solving from Nature, vol. 4193, pp. 613-622, 2006.

[174] Togelius, J. and Lucas, S. M., “Evolving robust and specialized car racing skills”, Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1187-1194, 2006.

[175] Togelius, J., “Optimization, Imitation and Innovation: Computational Intelligence and Games”, Ph.D. thesis, Department of Computing and Electronic Systems, University of Essex, UK, 2007.

[176] Togelius, J., De Nardi, R. and Lucas, S. M., “Towards automatic personalized content creation for racing games”, IEEE Symposium on Computational Intelligence and Games, pp. 252-259, 2007.

[177] Togelius, J., Lucas, S. M. and De Nardi, R., “Computational Intelligence in Racing Games”, Advanced Intelligent Paradigms in Computer Games, vol. 71, pp. 39-69, 2007.

[178] Togelius, J. and Schmidhuber, J., “An Experiment in Automatic Game Design”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 111-118, 2008.

[179] Togelius, J. and Lucas, S. M., IEEE CEC 2007 Car Racing Competition, http://julian.togelius.com/cec2007competition/, retrieved on 18 August 2008.

[180] Togelius, J., Lucas, S., Ho, D. T., Garibaldi, J. M., Nakashima, T., Tan, C. H., Elhanany, I., Berant, S., Hingston, P., MacCallum, R. M., Haferlach, T., Gowrisankar, A. and Burrow, P., “The 2007 IEEE CEC simulated car racing competition”, Genetic Programming and Evolvable Machines, vol. 9, no. 4, pp. 295-329, 2008.

[181] Togelius, J., Karakovskiy, S. and Koutnik, J., “Super Mario Evolution”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 156-161, 2009.

[182] Tozour, P., “The evolution of game AI”, AI Game Programming Wisdom, pp. 3-15, Charles River Media, Inc, 2002.

[183] Turing, A., “Computing Machinery and Intelligence”, Mind, vol. 59, no. 236, pp. 433-460, 1950.

[184] Vaccaro, J. and Guest, C., "Automated Dynamic Planning and Execution for a Partially Observable Game Model: Tsunami City Search and Rescue", Proceedings of IEEE Congress on Evolutionary Computation, pp. 3686-3695, 2008.

[185] van der Werf, E. C. D., Winands, M. H. M., van den Herik, H. J. and Uiterwijk, J. W. H. M., "Learning to predict life and death from Go game records", International Journal of Information Sciences, vol. 175, no. 4, pp. 258-272, 2005.

[186] van Hoorn, N., Togelius, J. And Schmidhuber, J., "Hierarchical controller learning in a first-person shooter", Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 294-301, 2009.

[187] van Hoorn, N., Togelius, J., Wierstra, D. and Schmidhuber, J., "Robust player imitation using multiobjective evolution", Proceedings of the Congress on Evolutionary Computation, pp. 652-659, 2009.

[188] van Lankveld, G., Spronck, P. and Rauterberg, M., "Difficulty Scaling through Incongruity", Proceedings of the fourth International Artificial Intelligence and Interactive Digital Entertainment Conference, AAAI Press, pp. 228-229, 2008.

[189] van Lankveld, G., Spronck, P., van den Herik, H. J. And Rauterberg, M., "Incongruity-Based Adaptive Game Balancing", Advances in Computer Games, pp. 208-220, 2010.

[190] Wang, H., Gao, Y. and Chen, X., "RL-DOT: A Reinforcement Learning NPC Team for Playing Domination Games", IEEE Transactions on Computational Intelligence and AI in Games, vol. 2, no. 1, pp. 17-26, 2010.

[191] Williams, R. J., "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning", Machine Learning, vol. 8, pp. 229-256, 1992.

[192] Xu, S. and Zhang, M., "Data Mining - An Adaptive Neural Network Model for Financial Analysis", Proceedings of the third International Conference on Information Technology and Applications, pp. 336-340, 2005.

[193] Yannakakis, G. N. and Hallam, J., "Evolving Opponents for Interesting Interactive Computer Games", Proceedings of eighth International Conference on the Simulation of Adaptive Behavior, pp. 499-508, 2004.

[194] Yannakakis, G. N., "AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation", Ph.D. thesis, University of Edinburg, 2005.

[195] Yannakakis, G. N. and Hallam, J., “A Generic Approach for Generating Interesting Interactive Pac-Man Opponents”, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 94-101, 2005.

[196] Yannakakis, G. N. and Hallam, J., “Capturing Player Enjoyment in Computer Games”, Studies in Computational Intelligence, vol. 71, pp. 175-201, 2007.

[197] Yannakakis, G. N., “How to Model and Augment Player Satisfaction: A Review”, Proceedings of the first Workshop on Child, Computer and Interaction, 2008.

[198] Yannakakis, G. N., Hallam, J. and Lund, H. H., “Entertainment capture through heart activity in physical interactive playgrounds”, User Modeling and User-Adapted Interaction, vol. 18, pp. 207-243, 2008.

[199] Yannakakis, G. N. and Hallam, J., “Real-time Game Adaptation for Optimizing Player Satisfaction”, IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 2, pp. 121-133, 2009.

[200] Yao., X., “Evolving artificial neural networks”, Proceedings of the IEEE, vol. 87, no. 9, pp. 1423-1447, 1999.

[201] Yong, C. H. and Miikkulainen, R., “Coevolution of Role-Based Cooperation in Multi-Agent Systems”, Technical Report AI07-338, Department of Computer Science, University of Texas, Austin, 2007.

[202] Zitzler, E., Laumanns, M. and Thiele, L., “SPEA2: improving the strength Pareto evolutionary algorithm,” Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, 2001.