Artif Intell Rev (2008) 29:123–161
DOI 10.1007/s10462-009-9112-y

Machine learning in digital games: a survey

Leo Galway · Darryl Charles · Michaela Black

Published online: 5 August 2009
© Springer Science+Business Media B.V. 2009

Abstract Artificial intelligence for digital games constitutes the implementation of a set of algorithms and techniques from both traditional and modern artificial intelligence in order to provide solutions to a range of game dependent problems. However, the majority of current approaches lead to predefined, static and predictable game agent responses, with no ability to adjust during game-play to the behaviour or playing style of the player. Machine learning techniques provide a way to improve the behavioural dynamics of computer controlled game agents by facilitating the automated generation and selection of behaviours, thus enhancing the capabilities of digital game artificial intelligence and providing the opportunity to create more engaging and entertaining game-play experiences. This paper provides a survey of the current state of academic machine learning research for digital game environments, with respect to the use of techniques from neural networks, evolutionary computation and reinforcement learning for game agent control.

Keywords Machine learning · Computational intelligence · Digital games · Game AI

1 Introduction

From the 1950s to present day, the development of game-playing programs has been a major focus of research for the domain of artificial intelligence (AI). Researchers were initially captivated by traditional board and card games, such as Chess, Checkers, Othello, Backgammon and Poker, with eventual successes being achieved in the creation of programs capable of

L. Galway (✉) · D. Charles · M. Black
School of Computing and Information Engineering, Faculty of Engineering, University of Ulster, Cromore Road, Coleraine, Co. Londonderry, BT52 1SA, UK
e-mail: galway-l1@email.ulster.ac.uk

D. Charles
e-mail: dk.charles@ulster.ac.uk

M. Black
e-mail: mm.black@ulster.ac.uk

competing with human opponents, in some cases, at world-class level. Pioneering work by Samuel (1959, 1967) into the development of a Checkers playing program, in particular the use of symbolic, search-based AI, automated feature selection and an embryonic version of reinforcement learning, underpins much subsequent machine learning and symbolic AI research conducted, especially within the domain of game-playing programs (Schaeffer 2000; Fürnkranz 2001; Schaeffer and Van den Herik 2002; Lucas and Kendall 2006; Miikkulainen et al. 2006). Initial research focused on *perfect-information* games (e.g. Chess, Checkers, Othello), requiring the development of game-playing solutions within large, completely specified, state-spaces, with later research efforts extending the application domain to the use of *imperfect-information* and *stochastic* games (e.g. Backgammon, Poker), thus expanding the challenge to include state-spaces with hidden states, probabilistic game-play and multiple opponents (Schaeffer and Van den Herik 2002; Lucas and Kendall 2006). By the end of the 1990s a number of milestones had been achieved, starting with the defeat of World Checkers Champion Marion Tinsley in 1994 by the program *Chinook* (Schaeffer 1997, 2000). Further successes followed in 1997 with World Chess Champion Garry Kasparov being defeated in an exhibition match by *Deep Blue* (Campbell et al. 2002; Hsu 2002), and the defeat of World Othello Champion Takeshi Murakami in an exhibition match by the program *Logistello* (Buro 1997). Similarly, a degree of success has been achieved using the game of Backgammon, with the program *TD-Gammon 3.0* (Tesauro 1995) displaying a convincing performance against the World Backgammon Champion Malcolm Davis in a series of matches during the 1998 American Association for AI's Hall of Champions game-playing exhibition. Proficient game-playing programs have also been developed for the game of Poker, notably *Loki* (Billings et al. 2002), which is capable of providing an intermediate-level challenge to human opponents, yet is still not quite able to play at world-class level (Schaeffer 2000; Lucas and Kendall 2006). For a substantial review and detailed discussion of traditional game-playing programs, please refer to Schaeffer (2000) and Fürnkranz (2001).

The development of game-playing programs for traditional board and card games has been fundamental to the growth of AI research, generating a wealth of innovative AI and machine learning techniques. Concurrently, the rise of low-cost, high performance home computing and subsequent growth of the computer games industry has led to the emergence of interactive computer games (also referred to as *video games* or *digital games*) with the creation of entirely new forms of game-play and corresponding genres of games. Representing a departure from traditional games, digital games make use of increasingly complex and detailed virtual environments, often incorporating human controlled protagonists and many computer controlled opponents (referred to as *game agents* and *opponent agents*, respectively). These agents require an ever more realistic and believable set of behaviours for a game engine's AI sub-system (*game AI*) to generate (Laird and Van Lent 1999; Schaeffer and Van den Herik 2002; Lucas and Kendall 2006; Miikkulainen et al. 2006). With a wide range of potential objectives and features associated with game-play, as indicated by the plethora of genres arising from digital games, such as 'First-Person Shooter Games', 'Action-Adventure Games', 'Role-Playing Games', 'Strategy Games', 'Simulation Games' (also known as 'God Games') and 'Sports Games', the specifications and requirements for a game AI typically include both genre-specific goals and general low-level behaviours (Laird and Van Lent 2001; Schaeffer and Van den Herik 2002). By utilising a percentage of the overall computational resources used within a game, the game AI facilitates the autonomous selection of behaviours for game agents through game specific perception, navigation and decision making sub-systems (Woodcock 2001, 2002; Tozour 2002; Charles 2003; Schwab 2004). Both traditional and modern AI techniques have been adopted by game developers and incorporated into game AI, including the use of algorithms such as A* for path-planning (Schwab 2004;

Buckland 2005; Miikkulainen et al. 2006), Finite State Machines (FSM) and neural networks for behavioural control (Charles and McGlinchey 2004; Buckland 2005), rule-based systems and hierarchies of task-networks for goal planning (Laird and Van Lent 1999; Hoang et al. 2005; Orkin 2005; Gorniak and Davis 2007), and evolutionary methods for both game engine parameter tuning and testing (Schwab 2004; Baekkelund 2006; Lucas and Kendall 2006). However, the majority of existing game AI implementations are primarily constrained to the production of predefined decision making and control systems, often leading to predictable and static game agent responses (Woodcock 1999; Charles 2003; Charles and McGlinchey 2004; Miikkulainen et al. 2006).

Typically the amount of computational resources allocated to game AI is game dependent, ranging from 5 to 60% of CPU utilisation, with turn-based strategy games receiving almost 100% of CPU resources when required (Woodcock 1998, 2000, 2001, 2002). With each successive generation of gaming hardware the amount of computational resources dedicated to game AI processing increases (Laird and Van Lent 1999, 2001; Woodcock 2002; Charles 2003; Schwab 2004). Coupled with an improved understanding of game AI techniques over successive generations of games, leading to a more thorough approach to game AI design and efficient implementation strategies, the quality of game AI has improved, further increasing the availability of CPU resources (Pottinger 2000; Woodcock 2000, 2001, 2002). With the increasing expectations and demands of game players for better, more believable game agents, the importance of game AI has gained widespread acceptance throughout the games industry as a differentiator and catalyst for the success or failure of products (Laird and Van Lent 1999, 2001; Woodcock 2002; Charles 2003; Schwab 2004). Such a desire for improved game AI is mirrored by the proliferation and adoption of middleware AI tools, such as *Kynapse,*[1] *AI.implant*[2] and *xaitEngine*.[3] Likewise, custom hardware solutions, such as the *Intia Processor*,[4] have also been developed for the acceleration of game AI techniques.

Attempts to incorporate machine learning into commercial game AI have been primarily restricted to the use of learning and optimisation techniques during game development and between periods of game-play, known as *offline* learning [also referred to as *out-game* learning (Stanley et al. 2005a)]. For example, offline learning techniques has been used for game AI parameter tuning in the game *Re-Volt*[5] and for the development of opponent agents in the games *Colin McRae Rally 2*[6] and *Forza Motorsport*.[7] Conversely, performing learning in real-time during game-play, known as *online* learning [also referred to as *in-game* learning (Stanley et al. 2005a)], has only been attempted in a handful of commercial digital games, including *Creatures*[8] and *Black and White,*[9] however, explicit learning controlled by the player is the key focus of the game design and game-play for these particular games (Grand et al. 1997; Woodcock 1999, 2001, 2002; Evans 2001; Manslow 2002; Charles 2003; Togelius and Lucas 2006). Through the use of online learning game agents may be enhanced with a capability to dynamically learn from mistakes, player strategies and game-play behaviours in real-time, thus providing a more engaging and entertaining game-play experience. However,

[1] Kynogon SA.

[2] Engenuity Technologies, Inc.

[3] xaitment, GmbH.

[4] AIseek, Ltd.

[5] Acclaim Entertainment

[6] Codemasters Software Company, Ltd.

[7] Microsoft Game Studios

[8] Cyberlife Technology, Ltd.

[9] Lionhead Studios.

the use of online learning raises a number of issues and concerns regarding the design and development of an appropriate machine learning algorithm. Subsequently, the integration of online learning within digital games also gives rise to issues for both game design and development (Woodcock 1999; Manslow 2002; Charles 2003; Baekkelund 2006; Lucas and Kendall 2006; Miikkulainen et al. 2006).

The aim of this paper is to provide a review of the key machine learning techniques currently used within academic approaches for the integration of learning within game AI. The following section provides an overview of the issues and constraints associated with the use of online learning within digital game environments. Section 3 presents a survey of the current academic digital game research literature, focusing on the use of methods from the computational intelligence domain, in particular the techniques of neural networks, evolutionary methods and reinforcement learning, utilised for both the online and the offline generation of game agent controllers. This will be followed by a summary of the literature reviewed together with conclusions drawn from the analysis of the literature in Sect. 4.

2 Issues and constraints with learning in digital game environments

This section begins with a discussion of the issues and constraints that arise from the use of learning within digital games before moving on to present a comprehensive review of the key machine learning approaches applied within the academic digital game research literature. Such issues will be discussed in terms of the constraints imposed on the learning algorithm by the digital game environment, the issues associated with the incorporation of learning into digital game AI, and the game design and development issues that arise from the use of a learning mechanism.

2.1 Digital game environment related issues

Research has shown that a player's interest in a game can be partially considered as a product of interactions with game agents within a game environment, where the game environment is characteristically nondeterministic and dynamic, containing multiple game agents with potentially discontinuous inputs (Duan et al. 2002; Kirby 2004; Yannakakis et al. 2004; Yannakakis and Hallam 2004). Due to the real-time nature of digital games, game agents are required to react quickly to any changes in the game environment (Maes 1995; Laird and Van Lent 1999; Lucas and Kendall 2006; Miikkulainen et al. 2006). The potential use of discontinuous inputs to game agents, implied by their typical short-term lifespan and ability to move around the game environment, imposes a requirement that any learning technique used should be able to make use of potentially varying input features. Digital games commonly contain a large number of internal *states*, with a corresponding set of actions associated with each state. As such, the *state-action space* manipulated by a game AI is typically high dimensional, therefore the use of exhaustive search techniques, in order to determine a suitable action for a game agent, may not be feasible during learning, particularly if an online learning mechanism is to be implemented (Blumberg et al. 2002; Stanley et al. 2005a,b).

2.2 Machine learning related issues

A major concern with the use of learning in digital games is the requirement for computational efficiency during both the *learning* and *classification* phases of the learning mechanism.

Any chosen learning technique should be capable of performing learning and classification within a specific period of time, using a limited set of resources, as defined by the amount of game AI resources available for the particular game genre (Charles and McGlinchey 2004; Kirby 2004; Schwab 2004; Baekkelund 2006). However, by incorporating domain specific knowledge into both the initial state of the learning algorithm and the learning process, improvements may be gained in the learning performance and classification capability of the algorithm (Manslow 2002; Baekkelund 2006). The choice of knowledge representation scheme used by a learning algorithm places a constraint on the capabilities of the learning technique as it implicitly defines the space of possible functions the learning algorithm can represent and subsequently learn. Through the use of a high-level, symbolic knowledge representation scheme, the manipulation of knowledge by the game AI will be straightforward and visible to game designers and developers. However, a high-level scheme may only be considered appropriate when the computational expense incurred in obtaining and manipulating the knowledge is low enough to allow for its efficient and effective use (Kirby 2004). An issue that arises as a consequence of learning is the problem of *over-fitting*; within digital games this may occur if a game agent has learned to adapt its performance according to a very specific set of states from the game environment and remains unable to perform generalisation, resulting in poor performance when new game states are encountered (Manslow 2002; Kirby 2004). Although the use of a set of validation examples is commonly used within applied machine learning to reduce over-fitting, this approach may not be suitable within a game environment due to the increase in time taken to learn a validation set of examples and the problems associated with obtaining a validation set when a limited amount of training data is available. Correspondingly, the selection of a set of training examples for a game AI learning task can be difficult to attain, particularly in game genres where a useful example may involve a sequence of actions with delayed or compound results. As such, a suitable learning technique may require the ability to derive training examples from player generated examples or the capability to learn without explicit targets (Kirby 2004; Lucas and Kendall 2006; Miikkulainen et al. 2006).

2.3 Game design and development related issues

One of the major game industry concerns regarding the use of learning within commercial games is the unpredictable nature of learning, potentially giving rise to the generation of unorthodox game agent behaviours. In particular, as online learning occurs during game-play, after a game has been shipped, this leaves game developers with no further control over the learning process, and as such, incorrect game agent behaviours learned may lead to an unconvincing and unsatisfactory game-play experience. Subsequently, there exists a requirement for a set of game-specific, developer defined constraints over the learning process in order to ensure that game agent behaviours learned do not cause deterioration in game-play or degradation of the overall game experience for the player (Woodcock 1999, 2002; Charles and McGlinchey 2004; Stanley et al. 2005a; Miikkulainen et al. 2006). Without such constraints, inappropriate or inconsistent game agent behaviours which occur through the use of learning may also give rise to increased costs in development time and resources required for bug reproduction, fixing and game testing. However, the use of learning may potentially reduce the resources associated with the development of game agent behaviours and may also be used to determine flaws and inconsistencies within both the game engine and game agent behaviours (Woodcock 1999, 2002; Schwab 2004; Baekkelund 2006; Lucas and Kendall 2006). Game agents should behave appropriately within the context of the game

environment, learning behaviours that are neither mechanistic nor predictable and remain consistent throughout the duration of the game. Likewise, learned game agent behaviours should be visible, believable and interesting to the player, rather than simply exhibiting *optimal* behaviours (Maes 1995; Laird and Van Lent 1999; Baekkelund 2006; Lucas and Kendall 2006; Miikkulainen et al. 2006; Togelius et al. 2007a). As the number of game agents used within a variety of game genres continues to increase with successive generations of games, one requirement that may be imposed on the use of any learning algorithm within game AI is the need for effective and efficient scalability. It is preferable that the learning technique used should be scalable to an arbitrary or predefined number of game agents. At the very least, steps should be taken to ensure that any active game agents in the immediate vicinity of the player perform sufficient learning based on their interactions or perception of the player. The degree of visibility that the learning process provides to game designers and developers should also be taken into consideration. In order for designers and developers to effectively tune the game AI, without the aid of offline learning, it is necessary that designers are capable of understanding the learning process, therefore the use of a meaningful knowledge representation scheme is highly desirable (Baekkelund 2006). Subsequently, it is advantageous for the development process if the chosen learning technique is simple or straightforward to implement and maintain for a range of game AI learning tasks (Laird and Van Lent 1999; Baekkelund 2006).

3 Machine learning techniques within digital game research

The following section provides a comprehensive review of the key machine learning techniques reported in academic digital game research literature. Although a wide variety of machine learning techniques exist, the focus of this review will be on three key techniques; neural networks, evolutionary methods and reinforcement learning, due to the ever increasing body of literature featuring the use of such techniques within digital game environments. Digital games are rapidly becoming a viable application domain for machine learning research, as exemplified by the growth of game-related conferences, such as the IEEE Symposia on Computational Intelligence in Games (CIG), the annual AAAI Artificial Intelligence for Interactive Digital Entertainment (AIIDE) conference, and both Eurosis' GAME-ON and the University of Wolverhampton's CGAMES series of conferences. With the proliferation of published machine learning related papers, encompassing research into a wide range of aspects of game AI creation, this review aims to concentrate on a subset of the application domain, specifically the generation of controllers for game agent behaviours. Each subsection will detail the use of a particular machine learning technique or approach, as reported within the academic research literature. Further summary and analysis will then be presented in Sect. 4.

3.1 Neural network approaches to learning in digital games

Neural networks present a class of learning models that are capable of providing a robust approach to learning discrete-valued, real-valued or vector-valued target functions. Consisting of a number of possible network topologies, incorporating a variety of single and multi-layered architectures containing a choice of feed-forward, feedback and lateral weighted connections between neurons, neural network learning has been applied to a range of learning tasks including function approximation, classification, prediction and control (Haykin

1994). A variety of neural network approaches have been used within the academic digital game research literature for the generation of game agent controllers and as a network-based controller representation scheme. In the majority of cases, the Multi-Layer Perceptron (MLP) network architecture (Haykin 1994) is used, with learning being performed using either the backpropagation algorithm (Rumelhart 1989), the Levenberg–Marquardt variation of backpropagation (Hagan and Menhaj 1994), or, more commonly, the use of evolutionary methods (details of research using hybrid evolutionary approaches to network training will be presented within Sect. 3.2). In several cases, ensemble-based network approaches have been used, typically implemented using a number of MLP networks. Similarly, Self-Organising Map (SOM) (Kohonen 1982) approaches to game agent control have also been attempted. For both supervised and unsupervised learning methods, player modelling approaches, using datasets recorded during game-play by human players, have been predominantly used.

In research conducted by Yannakakis et al. (2003), a MLP network was used for the generation of a game agent controller within a multi-agent test bed, entitled *FlatLand*, in which game agents compete to attain randomly positioned targets while avoiding collisions with each other. The MLP network utilised a feature vector consisting of the coordinates of a number of the nearest opponent agents and the target point, centred on the game agent, together with the distance to the target point. Experiments were conducted to compare the performance of game agents when the connection weights of the MLP network were determined using either the Levenberg–Marquardt variation of the backpropagation algorithm, or evolutionary methods. Training examples were obtained using a game agent controller with a set of near-optimal behaviours, created using an artificial potential field designed specifically for the test bed. Evaluating the best performing controllers from both backpropagation and evolutionary learning, the resulting game agents were tested over a number of runs of the simulation, with multiple copies of the game agent being used in each simulation run. Results indicated that the controllers trained using backpropagation were less proficient, though required less computational processing than the controllers generated using evolutionary methods. The authors concluded that the reliance on training data, along with the complex nature of the learning task, proved to be a limitation of the use of backpropagation for training the game agent controllers (Yannakakis et al. 2003).

Artificially created datasets were also used to train a MLP network in the research performed by MacNamee and Cunningham (2003). In this work, the μ-SIC system was presented, as a component of the Proactive Persistent Agent (PPA) architecture (MacNamee and Cunningham 2001), in which the MLP network was used to provide a basis for the behaviour of game agents when performing interactions with one another. Based on the use of a personality model, sets of values representing the personality and mood of a game agent with which an interaction is to take place, along with a set of values representing the relationship between the two game agents, were used as a feature vector to the MLP network in order to determine the type of interaction that occurs. Training was performed offline using the backpropagation algorithm with a set of artificially generated examples, due to the lack of an existing database of examples of social agent interactions. Evaluation was performed using an example simulation environment developed using the PPA system, with the μ-SIC system showing consistent performance for a wide range of game agent interactions that developed and changed over the course of the simulation. However, the authors have indicated that although the use of a trained MLP network is suitable for real-time use, training using backpropagation was found to be time consuming, therefore unsuitable for effective use as an online learning mechanism (MacNamee and Cunningham 2003).

In the research performed by Geisler (2004), a player modelling approach was used for the generation of game agent behaviours within the first-person shooter game *Soldier of*

Fortune 2.[10] Investigating the use of both a single MLP network and an ensemble of MLP networks, game agent controllers were trained using datasets generated from game-players, in an attempt to model the behaviours of players and to subsequently produce a network-based game agent controller capable of providing a subset of behaviours comparable to those of a human player. Utilising a feature vector based on spatial information about opponent agents and goals, specified relative to the game agent, backpropagation was used during network training for both the single network controller and the ensemble-based controller, in conjunction with the use of *bagging* (Breiman 1996) and *boosting* (Freund and Schapire 1996) techniques. During validation of the network controller's classification performance on the learning task, the controller network with the best performance was found to be an ensemble of four MLP networks which made use of boosting techniques during training. When the trained controller was integrated into the digital game, the author observed that the corresponding game agent displayed certain behaviours similar to a human player. However, it has also been pointed out that using a player modelling approach with data generated from a poorly performing player will result in a poorly performing game agent (Geisler 2004).

Bagging and boosting techniques, in conjunction with a player modelling approach, were also investigated in the research of Chaperot and Fyfe (2006), in which a MLP network was used for the control of a game agent within the racing game *Motocross The Force*. Investigating neural network training, the connection weights for both a single MLP network and an ensemble of MLP networks were determined during experiments using the backpropagation algorithm, with bagging and boosting techniques being utilised in an attempt to improve the algorithm's performance. The resulting controller networks were compared to the performance of a network that had been trained using evolutionary methods. By using a system of waypoints evenly situated around the track, the feature vector of the MLP network controller made use of waypoint space for the position, direction and velocity of the game agent, coupled with the height and position of the centre of the track, relative to the game agent. Pre-recorded player data from a number of different tracks was used to provide a set of training examples for the backpropagation algorithm. When bagging was used the performance and generalisation capability of the resulting controller were not found to be particularly good, in comparison to a controller trained using boosting or evolutionary methods. Training was also found to be inefficient in terms of the processing time and the computational resources used when bagging was performed (Chaperot and Fyfe 2006). In related research conducted by Togelius et al. (2007b) into the evolution of racing tracks (please see Sect. 3.2.5 for further details), a MLP network, trained using backpropagation on a set of player-generated examples, was initially used to obtain a player model for a simulated racing car controller. However, the resulting controller was reported to have performed extremely poorly. The authors of this work suggested this was not due to the use of the supervised learning approach, rather the datasets used during training did not contain sufficient examples of incorrect behaviours and so the learned controller was unable to recover from mistakes (Togelius et al. 2007b).

Both player generated training examples and automatically generated training examples were used in the research conducted by Bryant and Miikkulainen (2006a). In the work carried out, a MLP network representation was used within the Adaptive Team of Agents (ATA) multi-agent controller architecture (Bryant and Miikkulainen 2003). By creating a team of homogeneous agents with identical control policies, where each agent in the team corresponds to an individual game agent represented and controlled by an identical MLP network, an ATA controller for multiple game agents may be induced by the game agents simultaneously attempting to achieve separate, distinct goals encapsulated within the game

[10] Raven Software.

environment. In order to evaluate the ATA controller architecture, Bryant and Miikkulainen (2003) developed a discrete-state, turn-based strategy game, *Legion-I*, as a multi-agent learning task, which pits a number of game agents (*legions*) against a number of pre-programmed opponent agents (*barbarians*) in a competition for the possession of unit areas of the game map. For the game agents to succeed, they must cooperate to minimise the destruction incurred by the random invasion of large numbers of opponent agents. Control for individual game agents was performed using a MLP network; the same network being used in turn for each game agent to guarantee the ATA assumption that all game agents made use of an identical control policy. In the work carried out by Bryant and Miikkulainen (2006a), the MLP network used by the ATA controller was trained using backpropagation during a series of experiments conducted in order to investigate the utilisation of game agent sensor symmetries and game environment symmetries during network training. Using a modified version of the *Legion-I* game, entitled *Legion-II*, to allow for reflection and rotation of the game environment by the game agent's sensors, sets of human generated training examples were obtained, where each move recorded a set of game agent sensor inputs together with the corresponding action output. Further sets of examples were automatically created by applying reflection, rotation and a combination of both reflection and rotation transformations to the values contained in the original dataset, resulting in four overall datasets containing an increasing number of training examples, corresponding to the original dataset plus a dataset for each of the transformations used. Training was repeatedly performed using backpropagation on each of the datasets, with validation being periodically conducted on randomly generated test games. The best performing networks were then tested on a number of randomly generated test games, using the same test games to evaluate each of the selected networks. Results indicated that the best game agent performance and consistency of game agent behaviours was observed when the training set contained both reflection and rotation transformations. Correspondingly, although the authors have shown that the improved performance may be attributed to an increased number of training examples in the automatically generated datasets, the improvement in behavioural consistency was primarily due to the inclusion of examples containing symmetrical transformations (Bryant and Miikkulainen 2006a).

The use of pre-recorded player data for network training has also been used in the work of Bauckhage and Thurau (2004), in which a Mixture of Experts (MOE) architecture (Jacobs 1991), consisting of three MLP networks as the *expert* networks and another MLP network as a *gating* network, were used for context-sensitive game agent control within the first-person shooter game *Quake II*.[11] Using a player-generated dataset, with feature vectors containing spatial information about the game agent and a single opponent agent, the MOE network was trained using backpropagation in order to learn a control mechanism for selecting from three characteristically distinct weapons during game-play. When the resulting network controller was evaluated during game-play, the authors observed that the game agent controller was able to successfully switch weapons based on the current context of the game-play. Similarly, the authors also found that the game agent reproduced the targeting behaviours and shooting frequencies associated with each of the three weapons used, in a manner that was similar to the way in which the weapons were used by the player when the training examples were obtained (Bauckhage and Thurau 2004).

Using a player modelling approach, a SOM has been used as a game agent controller for the arcade game *Pong*[12] in the research carried out by McGlinchey (2003). By utilising

[11] Id Software, Inc.

[12] Atari Interactive.

pre-recorded player data, training with a low computational cost was achieved through the use of localised neighbourhoods, however, the resulting game agent controller was found to perform less than satisfactorily, resulting in erratic movements during game-play. The author suggested this was possibly due to quantisation being performed on the training examples. As in Geisler (2004), McGlinchey (2003) has also pointed out that the use of training data from a single player's pre-recorded game-play may be unsuitable for use in online learning due to sub-standard behaviours being learned if inappropriate behaviours are recorded in the training data. Thurau et al. (2003) have also made use of a SOM to cluster pre-recorded training data from the first-person shooter game *Quake II*, with the resulting individual clusters being used to train a number of MLP networks in order to generate controllers for a game agent's view-angle and velocity. Subsequently, separate datasets were generated for training, one dataset containing examples of player movement behaviours and the other dataset containing examples of player movement behaviours together with aiming behaviours. The Levenberg–Marquardt variation of backpropagation was used to train the MLP networks. In experiments conducted into the topology of the SOM, along with the use of either a one or two MLP networks, it was found that increasing the size of the SOM helped reduce the network training error and a single MLP network was capable of successfully learning game agent velocity adjustments from the dataset containing player movements only. When the dataset containing both movement and aiming behaviours was used, results indicated that two MLP networks were required in order to learn game agent velocity control and game agent view-angle control, with little improvement being shown when more than two neurons were used in the SOM. The authors have also reported that the resulting game agent controller demonstrated suitable and appropriate behaviours when evaluated during game-play.

3.2 Evolutionary approaches to learning in digital games

Evolutionary computation comprises a class of optimisation techniques which utilise simulated evolutionary processes in order to search through a problem space for an optimal solution. By representing the solution to a problem in genetic terms, a population of potential solutions is generated and maintained through the use of genetic operators. Each potential solution is encoded as a genotype, represented by a chromosome string, which is decoded into its corresponding phenotype form in order to enable the fitness of the genotype to be evaluated. Evolutionary methods enable the process of learning to be considered as a special case of optimisation, whereby the learning process attempts to discover an optimal solution based on the fitness of individuals within a population of solutions (Holland 1975; Goldberg 1989). Academic digital game research has applied a range of techniques from evolutionary computation to the generation of game agent controllers. Evolutionary Algorithm (EA) techniques have been used for both offline and online learning, including Genetic Algorithms (GA) (Holland 1975; Goldberg 1989), Evolutionary Strategies (ES) (Beyer 2001) and Genetic Programming (GP) (Koza 1992). Commonly, the use of evolutionary methods involves the hybridisation of evolutionary computation with other machine learning and AI techniques, such as neural networks, in an attempt to find more efficient and robust solutions to game-specific learning tasks and in order to help automate the learning process. The following subsection will be split into subsections corresponding to the main game agent controller representations and evolutionary techniques used within the research reviewed.

3.2.1 Evolutionary optimisation of rule-based game agent controller representations

In the work of Ponsen and Spronck (2004) *Dynamic Scripting* techniques have been used, in which an EA has been utilised to generate domain knowledge, for inclusion in a rule-base of tactics, used by game agents within the real-time strategy game *Wargus*. Representing a complete set of scripted behaviours for a game agent within each chromosome of the evolving population, individual genes of a chromosome correspond to one of four rules for tactical game-play together with a set of parameter values associated with each rule. A population of chromosomes was initialised with randomly generated values and evolved using tournament selection and a set of custom genetic operators. Performing evolution offline against pre-scripted game agents, the authors found that the resulting game agent scripts were capable of creating both effective and robust game agent strategies, which can then be used to improve the domain knowledge used by the game AI, thus improving the corresponding game agent's performance (Ponsen and Spronck 2004). In an alternative EA approach to the evolution of a rule-base for game agent control, Gallagher and Ryan (2003) have made use of Population-Based Incremental Learning (PBIL) (Baluja 1994), in order to evolve the parameters of a set of probabilistic rules used to provide a game agent control strategy within a simplified version of the arcade game *Pac-Man*.[13] The game agent has been represented as a FSM with two-states, an *Explore* state and a *Retreat* state, with choices for moves in each state being selected using the set of probabilistic rules. Using a representation of the game environment based on the current orientation of the game agent, hand-coded rules were used to classify both the current location of the game agent, according to orientation and turn type, and the position of the opponent agent, relative to the game agent, resulting in a set of parameters which specify the behaviour of the game agent based on the probability of a move being made or a state transition occurring in relation to each rule. Experiments were conducted using PBIL to evolve the parameters for the game agent against an opponent agent with a predominantly deterministic strategy, which was then compared to game agents generated using hand-coded parameter values. The game agent created using PBIL showed a degree of stochastic behaviour and the set of parameter values generated converged to values similar to one of the hand-coded game agents, in which a strategy that prevented retreating in the Explore state was used. The performance of the hand-coded game agent, however, was reportedly better than that of the PBIL game agent (Gallagher and Ryan 2003).

Evolution of rule-bases for game agent control has also been investigated in an ongoing series of research performed by Parker et al. (2005a, 2006) and Parker and Parker (2006a, 2007b), which makes use of the air-combat game *Xpilot* as a test bed for their research. In work conducted by Parker et al. (2005a) a canonical GA was utilised to evolve a set of consequents and priorities for a rule-base used as a reactive game agent controller. Using a binary chromosome representation, roulette wheel selection, and both inter-gene crossover, to retain useful traits between generations, and intra-gene crossover, to maintain population diversity, the authors found that game agent behaviour within a single agent, single opponent environment improved substantially as evolution progressed. However, although multiple behaviours were evolved, the authors speculated that more complex behaviours could be evolved given a larger rule-base and increased chromosome length (Parker et al. 2005a). Using a similar learning task, research into the evolution of decision and action parameters for a hand-coded, rule-based game agent controller was further performed by Parker and Parker (2007b). Again, a canonical GA was employed to evolve the parameters for the game agent's rule-base against a hand-coded opponent agent, with an individual's fitness

[13] Namco, Ltd.

determined by the number of game frames survived together with a bonus for killing opponent agents. Although the authors reported that the fitness scores obtained showed a large degree of variation, attributed to noisy fitness evaluation due to changing starting positions of both the game agent and the opponent agent during each generation of evolution, the average and individual fitness scores increased as evolution progressed, leading to game agents that were competitive with human opponents and displayed emergent, complex behaviours (Parker and Parker 2007b).

Further developing the research by Parker et al. (2005a) into the evolution of consequents for a hand-coded rule-base game agent controller, Parker et al. (2006) applied a Cyclic Genetic Algorithm (CGA) approach (Parker and Rawlins 1996), featuring a single loop encoded using a binary representation for each chromosome, to the evolution of a hand-coded rule-base for reactive game agent behaviours. Using a Steady State GA (DeJong and Sarma 1992) with first-in first-out deletion and roulette wheel section to affect the use of a CGA, the authors have reported that the resulting controllers outperformed previously generated rule-based controllers on the same learning task. However, the requirement for a hand-coded set of rule antecedents, generated using expert knowledge of the learning task, presented a potential limitation of this approach (Parker et al. 2006). Continuing this research, Parker and Parker (2006a) attempted the use of a competitive co-evolutionary approach to the generation of a game agent controller using a multi-loop CGA, where each loop represented a set of game related antecedents and corresponding consequents. By developing a customised, parallel learning environment for the evolution of *Xpilot* agents, entitled *The Core*, a single population of game agents were co-evolved in a competitive manner alongside the evolution of individual game agents within the population. During the evolutionary process the chromosomes of successful individual game agents were recombined with the mutated chromosomes of weaker, unsuccessful opponent agents, thus the environment provided both selection and crossover operations that attempted to perpetuate useful genetic traits to the entire population. In a similar manner, fitness was considered as a by-product of the rules and setup of the game environment, rather than an explicitly predefined fitness function (i.e. fitness reflected the current capabilities of an individual game agent when tested within the *The Core*). Using a standardised set of opponent agents, the authors tested randomly selected game agents evolved in The Core over a period of time. Results indicated that the performance of the game agent increased over time, which was corroborated by periodic visual inspection of the game agent's behaviours by the authors (Parker and Parker 2006a).

A co-evolutionary approach has also been used for the evolution of a number of rule-based opponent agents in experiments conducted by Demasi and Cruz (2003). Using a multi-agent action game as a test bed, featuring a single player competing against a number of opponent agents, co-evolution was performed in order to adapt the control strategies of the opponent agents to the game-playing ability of the player in real-time. By representing the four possible actions corresponding to each of the rules contained in the rule-base within an individual binary chromosome for each opponent agent, evolution was used in order to determine an optimal set of actions for each rule that may be used to control an opponent agent's behaviour during game-play. In a number of co-evolution experiments, individual game agent behaviours were evolved using separate subspecies within a GA, utilising roulette wheel selection and a cooperative fitness function between subspecies in which the performance of both the opponent agent and the player were taken into account. Population diversity was maintained by using a replacement strategy that introduced randomly generated individuals for 10% of each generation of the evolving population. When evolution was performed against a player in real-time, the authors have reported that co-evolution was successful in evolving increasingly effective behaviours for the opponent agents using current game data. However, the

authors have also reported that online co-evolution was somewhat inefficient and suggested it may be more suited to games in which opponent agents are replaced with a high frequency, thus allowing the evolutionary process to be frequently performed in response to the request for a new opponent agent (Demasi and Cruz 2003).

3.2.2 Evolutionary optimisation of influence map-based game agent controller representations

Using an alternative approach to the representation of game agent controllers, Miles et al. (2007) have conducted experiments using competitive co-evolution with a representation scheme based on the use of Influence Maps (IM) (Tozour 2001). Focused on providing a real-time reaction to both players and opponent agents, strategic game agents have been evolved for use in a real-time naval combat simulator entitled *Lagoon*. Utilising a parallel Steady State GA to co-evolve populations of Influence Map-based Artificial Intelligence (IMAI) agents, each individual in a population represented all the parameters and coefficients for an IM Tree-based representation of game related resource allocations and tactical spatial reasoning. Extending previous research performed by Miles and Louis (2006), in which a GA was used to evolve both the allocation of resource units to game-related objectives and also to co-evolve the Influence Map Trees for tactical spatial reasoning, Miles et al. (2007) found the co-evolved game agents were more than capable of beating hand-coded opponents agents within a relatively small number of generations of evolution and eventually evolved to be capable of beating human opponents (Miles et al. 2007).

3.2.3 Evolutionary optimisation of program-based game agent controller representations

Using a standard implementation of GP, Koza (1992) presented a GP solution to the evolution of a program-based game agent controller for a task-prioritisation problem, using a simplified version of the arcade game *Pac-Man*. Employing a set of hand-coded primitive operators, defined in relation to the game agent, comprising operators for controlling the game agent's moves and a set of operators for performing distance calculations to features within the game environment, GP was used to evolve controllers against four hand-coded opponent agents that made use of a predominantly deterministic control strategy. Although a game agent controller was evolved which successfully navigated the game environment and was able to achieve the task of eating dots, the author reported that results obtained were inconclusive regarding the capability of the evolved game agent controller to exhibit generalised behaviours (Koza 1992).

The use of multi-objective fitness measures were investigated in the work of Agapitos et al. (2007b), which focused on the evolutionary generation of programs for game agent control in a single car, single track, point-to-point racing task within a RC car racing simulation. Using GP to evolve controller programs, multiple fitness metrics were used in order to increase evolutionary selection pressure towards the promotion and effective use of state variables and associated primitives for shared memory manipulation within the evolved programs, while at the same time encouraging the generation of proficient controllers for the learning task. Comparing programs evolved using the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) algorithm (Deb et al. 2002) with programs evolved using a generational GA incorporating elitism, tournament selection and a performance-based single-objective fitness measure, the authors found that programs in which the effective use of state variables had been evolved showed an increase in overall controller performance (Agapitos et al. 2007b).

3.2.4 Genetic algorithm optimisation of connection weights for neural network-based game agent controller representations

In Yannakakis et al. (2003, 2004) and Yannakakis and Hallam (2004) evolutionary methods have been used to determine the connection weights for neural network-based game agent controllers. As part of an investigation into the use of a MLP network game agent controller for the generation of complex game agent behaviours using the multi-agent test bed *FlatLand* (please refer to Sect. 3.1 for further details), Yannakakis et al. (2003) have utilised a GA with a real-valued chromosome representation, elitism selection and a randomly initialised population to evolve the connection weights of the controller. Using a fitness function that promotes the game agent objectives of obstacle-avoidance and target-achievement, the evolved controller exhibited more robust behaviours and performance improvements over equivalent neural network controllers trained using the back-propagation algorithm (Yannakakis et al. 2003). Extending previous work to the evolution of connection weights of MLP network controllers for multiple opponent agents, Yannakakis et al. (2004) and Yannakakis and Hallam (2004) made use of cooperative co-evolution within the predator/prey test bed *Dead End* and a simplified version of the arcade game *Pac-Man*, respectively. Using a fitness function based on the performance of individual game agents, a population of opponent agents was initially evolved offline against a set of fixed-strategy game agents in order to generate opponent agent controllers capable of performing complex teamwork behaviours. Subsequently, the chromosomes of the evolved controllers were used to seed a population of opponent agents for further online evolution. During online evolution the fitness function was based on the interactions between various fixed-strategy game agent controllers and the opponent agents in order to transform the initial population of homogeneous MLP network controllers into a set of heterogeneous controllers. For both offline and online evolution a GA was used, incorporating a real-valued chromosome representation and elitist selection. In the case of offline evolution, the initial population of chromosomes were randomly initialised. When performing online evolution, a significantly smaller number of simulation steps were used during evaluation, compared to the number of steps used during offline evolution. Similarly, the replacement strategy used differed between offline and online evolution, with all selected parent chromosomes being used to generate offspring during replacement in offline evolution and only the most-fit parent being used to generate a single offspring to replace the least-fit member of the population when online evolution was performed. In Yannakakis and Hallam (2004), a further step was introduced during online evolution in which a cloned offspring was also evaluated for a small number of simulation steps, using offline evolution, before replacing the least-fit member of the population, in order to reduce the possibility of mutated chromosomes reducing the performance of the evolving population (Yannakakis and Hallam 2004). By using the online evolutionary mechanism, starting with a population evolved offline against game agents with a range of fixed-strategies, the authors found that, for both *Dead End* and *Pac-Man*, the opponent agent performances were observed to rapidly adapt in real-time to the changing game agent behaviours (Yannakakis et al. 2004; Yannakakis and Hallam 2004).

As part of a comparative investigation into the performance of a MLP network game agent controller for the racing game *Motocross The Force*, Chaperot and Fyfe (2006) also made use of a GA to determine the connection weights for the network controller (again, please refer to Sect. 3.1 for further details). Utilising a real-valued chromosome representation scheme, the authors found that the creation of offspring during crossover was improved by randomly selecting from two crossover techniques; one in which random

crossover points are chosen and the value contained in the parents' chromosomes swapped, another which averages the values contained in both parents' chromosomes. The authors also reported that generating an initial population of MLP networks through the mutation of an existing pre-trained network, rather than through the use of an initial population of randomly generated networks, helped prevent the GA from removing beneficial features during recombination operations. In comparison with controllers trained using the backpropagation algorithm, the authors concluded that the evolved game agent controllers exhibited both a higher performance and more adaptable behaviours (Chaperot and Fyfe 2006).

The hybridisation of evolutionary methods and neural networks has also been investigated in the work of Parker et al. (2005b) and Parker and Parker (2006b, 2007a). Parker et al. (2005b) performed initial research into the use of GA to evolve the weights of a single-layer Perceptron controller for reactive game agent behaviours against a single hand-coded, rule-based opponent agent within the *Xpilot* game. Using a canonical GA, with a binary chromosome representation, roulette wheel selection and two-point crossover, the average fitness of both individual game agents and populations of game agents improved as the evolution progressed. It was found that using a feature vector based on a fixed set of coordinates from the game environment prohibited the learning of general behaviours rather the controller would only learn behaviours for specific locations of the game environment. In order to overcome this, feature vectors used as input to the controller were first transformed from a fixed coordinate system to coordinates relative to the current position of the game agent (Parker et al. 2005b). In Parker and Parker (2006b) experiments were conducted into the evolution of a modular network controller comprising three single-layer Perceptron networks, in order to control the game agent's bullet avoidance, shooting and attack navigation behaviours respectively, together with another single-layer Perceptron network for the aggregation of the three individual networks. Each of the three *behaviour* networks was evolved separately, utilising a GA to determine the connection weights of each network within a custom game environment representative of the individual learning task. The resulting best individual networks were combined to form the first layer of an overall two-layer network game agent controller, with the connection weights for the second layer of the overall network being dynamically determined by another single-layer network. Again, a GA was used to determine the connection weights of the *aggregation* network along with threshold values for the output layer of the overall controller network. When tested against a hand-coded opponent agent, the authors reported the multi-layer control network generated effective, aggressive game agent behaviours (Parker and Parker 2006b). Continuing their research into network-based game agent controllers, Parker and Parker (2007a) experimented with the use a distributed evolutionary method, the Queue GA (QGA) (Parker and Parker 2006c), to optimise the control structure and threshold-adjusted connection weights of a fixed topology MLP network game agent controller. By simultaneously evolving the choice of a predefined range of direct and comparative inputs to be used for the input layer of the MLP network, together with the connection weights and corresponding thresholds, the issues associated with the selection of a network's inputs and parameters were surmounted. The authors report that although the MLP controller did not produce as complex a set of behaviour as initially envisaged, the fitness of the evolving controllers, measured in terms of the length of survival and number of opponent agents destroyed, increased over the duration of the evolutionary process, resulting in game agent controllers that were both proficient and adequately adapted to the simplified *Xpilot* environment used (Parker and Parker 2007a).

3.2.5 *Evolutionary strategy optimisation of connection weights for neural network-based game agent controller representations*

In the work of Lucas (2004) a random mutation hill climber was used to determine the connection weights of a single-layer Perceptron controller for the artificial life simulation test bed *Cellz*. Symptomatic of randomly configured game settings, a high degree of variation was observed in the results obtained over a number of games, resulting in noisy fitness evaluation during evolution. In order to overcome such noisy fitness evaluations, the mean fitness was used, calculated over a number of games. Although simple controllers were successfully evolved, the use of a single-layer Perceptron was found to limit the capabilities of the controller, with hand-coded controllers exhibiting much better performance (Lucas 2004). Subsequent research by Lucas (2005) focused on the evolution of reactive game agent controllers for a modified version of the arcade game *Ms. Pac-Man*,[14] utilising an ES to optimise the connection weights of both Perceptron and MLP network based controllers. Using a graph-based representation of the game environment, calculations were performed prior to the evolutionary process to determine the shortest-path distances between all graph nodes, with the results stored in a look-up table to permit the efficient generation of feature vectors for the controller networks. The feature vectors used comprised the current location of a specified graph node, alongside a number of distance measures centred on the specified node. In order to generate a move for the game agent, the network controller was used to evaluate each potential move from the node corresponding to the game agent's current position, choosing a move in the direction of the node that received the best evaluation. Using an ES algorithm to determine the connection weights for both Perceptron and MLP network controllers, experiments were conducted in which controllers were evolved against opponent agents with deterministic and non-deterministic strategies. Results indicated that Perceptron controllers evolved against deterministic opponent agents performed poorly when tested in games against non-deterministic opponent agents. The author has pointed out that the use of non-determinism may permit more general game agent behaviours to be learned, however, it can also lead to noisy fitness evaluation during evolution, resulting in the need for a larger number of games to be played when determining an evolved controller's fitness. Subsequently, it was suggested that a suitable alternative is to increase to the population size of the ES. When using non-deterministic opponent agents, the best performing controller, in terms of the average score obtained, was a MLP controller evolved using a $(10+10)$ ES, which showed game agent behaviours that were more focused on eating the opponent agents than clearing the game level (Lucas 2005).

Similarly, Gallagher and Ledwich (2007) used an ES to evolve the connection weights of a MLP-network game agent controller within a simplified version of *Pac-Man*. In an attempt to make use of *raw* information from the game environment, the controller network's feature vector consisted of a number of *windows* centred on the game agent during game-play, containing binary representations of on-screen game information pertaining to the location of game features, including walls, dots and an opponent agent, together with counts of the remaining dots in each of the four quadrants of the game level. Initial experiments were conducted using a $(30+15)$ ES to evolve the connection weights of the network controller against a single opponent agent that made use of a deterministic strategy, with three 5×5 windows being used as part of the network's feature vector. The results showed that it was possible to use raw on-screen information to generate controllers that improve their performance during evolution. Due to the observed behaviours of the best performing game agents,

[14] Midway Games.

the authors have suggested that the more proficient controllers were more reactive to the input windows containing information about walls and dots. Further experiments were conducted to investigate the effect on the performance of controllers evolved against a non-deterministic opponent agent, with various changes being made to the experimental setup, including different sizes specified for the network's hidden layer, initial population and replacement strategy, together with the size of the input window used. Although learning was observed during the evolutionary process, the authors have reported that the performances of the evolved controllers were significantly lower than controllers evolved against a deterministic opponent agent, with results being inconclusive with regard to the effect on performance caused by the changed network and algorithmic parameters (Gallagher and Ledwich 2007).

In ongoing research conducted by Togelius and Lucas (2005); Togelius and Lucas (2006), Agapitos et al. (2007a), Lucas and Togelius (2007) and Togelius et al. (2007a,b), a variety of EA techniques have been used for the evolution of a range of game agent controller architectures, including MLP network-based controllers, for use in a RC car racing simulation. In the work of Togelius and Lucas (2005) an EA, which made use of truncation-based selection and an elitist replacement policy, was used for the evolution of a number of game agent controller architectures based on three categories of environmental state representation (simulated sensor information, Newtonian information and timing information). Controller architectures used included action-sequence lists, force fields and MLP networks, with the best evolved controller, consisting of a MLP network with simulated sensor inputs, being capable of marginally outperforming human competitors over point-to-point racing on a single track. The EA was used to evolve both the network connection weights alongside parameters for the game agent's wall sensors. The authors reported that higher fitness values were obtained during evolution when fixed game agent starting positions and orientations were used, leading to less noisy fitness evaluation and the subsequent generation of more robust controllers (Togelius and Lucas 2005). In an extension of this research, Togelius and Lucas (2006) have used an EA similar to a (50+50) ES in order to generate both generalised and specialised MLP-based controllers, capable of point-to-point racing on a variety of tracks and specific tracks, respectively. By incrementally evolving the connection weights for MLP networks containing two hidden layers, wherein a succession of tracks were individually added to the algorithm's fitness calculation as successful controllers were generated, a controller capable of generally proficient performance on a number of tracks was obtained. In order to obtain a specialised controller capable of high performance on a previously unseen track, the authors reported that the use of further evolution, using the controller with the best generalisation performance as an initial seed for the evolving population, successfully generated a track-specific controller capable of outperforming a human opponent (Togelius and Lucas 2006). Similar research into the evolution of generalised controllers has also been conducted by Agapitos et al. (2007a), in which a (50+50) ES was used to evolve a number of MLP network, recurrent network and GP controller representations, using a single objective fitness measure based on controller performance. For experiments conducted into the generalisation performance of evolved controllers using incremental fitness measures, similar to the work of Togelius and Lucas (2006), the authors found that both network-based and GP-based controllers were capable of generalisation, however, network-based controllers significantly outperformed GP-based controllers, though GP-based controllers took less time to evolve (Agapitos et al. 2007a).

Evolutionary strategies have also been used in the work of Lucas and Togelius (2007) and Togelius et al. (2007a,b). Building on the authors' previous work, Lucas and Togelius (2007) conducted research into the generation of controllers for single player point-to-point simulated RC car racing on a number of randomly generated tracks, utilising ES and reinforcement

learning to learn a range of direct, state-value and action-value based controllers with a variety of representations including both single-layer Perceptron and MLP networks. Using a (15 + 15) ES for the evolution of a network's connection weights, the authors found that the most proficient controllers were MLP networks used for game state evaluation. Subsequently, the authors have reported that, in general, the use of state-value based learning produced superior results to either direct or action-value based learning. In the research conducted it has also been reported that transformation of the raw game state information into an agent-centric feature vector, using expert knowledge of the problem domain, was required in order to obtain good game agent performance (Lucas and Togelius 2007). Within the research conducted, a (15 + 15) ES was also used to optimise the value function representation (i.e. look-up table or MLP) used by a reinforcement learning algorithm and in order to further evolve solutions previously learned using reinforcement learning. Proficient action-value based controllers were obtained for controllers used in a number of experiments conducted, with marginal improvements observed in the performance of controllers which made use of an evolved table-based value function representation, rather than controllers which made use of an evolved MLP network for value function representation. However, results from evolutionary learning using a controller generated using reinforcement learning in order to seed the initial population varied according to the learning task (please see Sect. 3.3 for a more detailed discussion of reinforcement learning approaches to game agent control) (Lucas and Togelius 2007).

Using a learning task similar to Lucas and Togelius (2007), the research performed by Togelius et al. (2007a) examined the use of both generational and steady-state, single and multi-population, co-evolution for the generation of a range of controllers for single and multi-player point-to-point simulated RC car racing. In this work, a number of control architectures were co-evolved, including MLP networks, Elman recurrent networks (Elman 1990), modular networks (consisting of a MLP network for decision making coupled with a recurrent network for game agent control), and a GP-based controller. For the single population, generational co-evolution, a standard ES was used, in which individuals in the population were evaluated against randomly selected individuals from the most-fit half of the population and the least-fit half of the population were replaced with a mutated copy of the most-fit half of the population during each generation of evolution. By contrast, in the multi-population generational co-evolution, individuals competed against the most-fit individuals from each population. For the experiments conducted using steady-state co-evolution, a modified *N-Strikes-Out* algorithm (Miconi and Channon 2006) was used. In the single population version, individuals of the population compete against each other and those that are defeated after N-strikes are replaced with a new individual. When used for multi-population co-evolution, competition between individuals was generalised to a number of populations and a *defeat factor* was introduced, which decayed the number of strikes for an individual based on the individual's absolute fitness, in order to overcome the potential replacement of highly fit individuals after a small number of competitions due to the use of a noisy fitness function. In terms of the best overall performing controller, the authors have reported that controllers based on a modular architecture, evolved using multi-population generational co-evolution, obtained a higher performance on both single and multi-player racing tasks. In particular, it was found that modular controllers for which the recurrent network component had already been pre-evolved to achieve reasonable performance would exhibit a higher performance than modular controllers evolved from scratch. In general, the authors found multi-population co-evolution produced better controllers than either single-population co-evolution or standard evolutionary methods (Togelius et al. 2007a).

In an alternative, novel approach to the use of evolutionary computation within simulated RC car racing, Togelius et al. (2007b) presented research on the automatic creation of racing tracks, tailored to the player, employing a modified ES to evolve a series of tracks using b-spline representations. By first evolving a neural network-based controller capable of generalised performance on a range of tracks, similar to the work presented in Togelius and Lucas (2006), an ES was utilised to further evolve the controller in order to indirectly model the playing style and performance of a human player, therefore necessitating the use of a multi-objective fitness function during the evolutionary process. Correspondingly, a multi-objective fitness function was also required during the evolution of bespoke tracks for the player-modelled controller, to ensure the customised tracks retained a number of key *entertainment features*, including both a suitable and varying amount of challenge. In order to overcome the need for multi-objective fitness measures, members of the population were selected during evolution using a non-linear combination of individual fitness functions, termed *Cascading Elitism*. The modified ES was used for both the further evolution of the player-modelled controller and the evolution of the customised tracks. Once the controller was satisfactorily obtained, it was subsequently used to test the evolved tracks (Togelius et al. 2007b).

3.2.6 Enforced sub-populations optimisation of connection weights for neural network-based game agent controller representations

An alternative co-evolutionary method, the Enforced Sub-Populations (ESP) algorithm (Gomez and Miikkulainen 1997) has been used in a series of experiments conducted by Bryant and Miikkulainen (2003, 2006b, 2007) for the optimisation of a neural network-based game agent controller, as part of the ATA multi-agent controller architecture, for use within the discrete turn-based strategy game test beds *Legion-I* and *Legion-II* (please refer to Sect. 3.1 for more details of the ATA architecture and *Legion* games). In the research presented, the evolution of a neural network's connection weights was performed using the ESP algorithm, in which separate sub-populations are used to directly encode the connection weights for individual neurons in the overall network, rather than encoding an entire set of weights within each sub-population. By maintaining a separate sub-population for each neuron in the network, crossover and mutation are carried out within these individual sub-populations, resulting in cooperative co-evolution between the sub-populations of a complete set of neurons for the overall network. In the work carried out by Bryant and Miikkulainen (2003), the turn-based strategy game *Legion-I* was used to evaluate the ATA controller architecture. The ESP algorithm was employed to generate the connection weights for a MLP network representation of the individual game agents within the ATA controller; the same network used in turn for each game agent to ensure all game agents utilised an identical control policy. Selecting the best networks obtained from a number of runs of the learning algorithm on independently generated sets of training and validation games, the performances of the resulting ATA controllers were evaluated both qualitatively, by observing the game agents during game-play, and quantitatively, by measuring the scores over a number of test games. The authors found that, for all networks tested, *purposeful* behaviours by the game agents were observed, alongside scores that indicated proficient performance. Subsequently, the authors concluded that the generation of ATA controllers, using the ESP algorithm, provided game agents with a capacity for producing adaptive behaviour (Bryant and Miikkulainen 2003).

Continuing this research, Bryant and Miikkulainen (2006b) proposed a method to enhance the game agent behaviours generated using evolved neural network controllers by allowing

for a degree of random behaviours to be used during network training. The technique, termed *stochastic sharpening*, decodes an evolved network's outputs during training using confidence-weighted decoding, in which a probability is determined for each output proportional to the output's activation level. Rather than decoding the network's outputs using a winner-take-all approach, where the output with the peak activation is chosen as the action to perform, an action is selected based on a pattern of confidence values that reflect the relative utilities of each output's associated actions. Subsequently, during evolution of a neural network's connection weights, patterns containing greater probabilities will signify behaviours that are more contextually appropriate, thus will be propagated throughout the evolving population. To evaluate the use of stochastic sharpening during the evolution of a network's connection weights and the use of stochastic sharpening when the evolved network controller is being used in-game, an ATA controller, with a MLP network game agent representation, was evolved using the ESP algorithm and used as a multi-game agent controller for the turn-based strategy game *Legion-II*. Comparing controllers using winner-take-all decoding during both training and testing against controllers using stochastic sharpening during training, and either winner-take-all decoding or stochastic sharpening during testing, the results indicated that convergence during learning with both methods was very similar, however, the performances of controllers that made use of stochastic sharpening during training, regardless of the decoding method used during testing, were improved by a statistically significant amount. The authors reported that the use of confidence-weighted decoding during testing also reduced the variance observed in the test results, which may reduce the amount of learning required in order to obtain consistent results. Introducing a controlled degree of randomness during the determination of output selection, networks trained using the stochastic sharpening method showed improved performance over those trained using standard winner-take-all decoding, independent of the amount of randomness introduced during either method used for decoding. As such, the authors concluded that the use of stochastic sharpening permits the use of stochastic behaviours for game agents with minor degradation to the game agent's overall performance, leading to the generation of more believable game agents (Bryant and Miikkulainen 2006b).

Further research using the *Legion-II* game, again making use of MLP network game agent controllers trained using the ESP algorithm, has been undertaken by Bryant and Miikkulainen (2007) which presented a method for inducing game agent behaviour policies from player generated examples during the evolutionary process. By incorporating an adaptation step into the evolutionary process, during which useful phenotypic traits are written back into the chromosome representation (also known as *Lamarckian evolution*), the evolutionary process may be guided by using examples of advantageous game agent behaviours. Using a set of examples of input vectors with corresponding actions for task specific behaviour policies, generated by a player aiming to perform specific objectives during a game, the ESP algorithm may be modified to perform Lamarckian evolution by using a single iteration of backpropagation during fitness evaluations, in order to incorporate the task specific examples into the chromosomes of the evolving population. Results from experiments conducted by performing task-specific objectives, using controllers generated using both the standard and modified ESP algorithm, along with controllers generated from the example data using backpropagation alone, have shown that the use of guided evolution permitted the generation of game agent behaviour policies which were closer to the task-specific player behaviours than when standard evolution or backpropagation were used. Similarly, the use of player generated examples to guide the evolutionary process has been shown to improve both the speed and accuracy with which rules, implicit in the target task-specific objectives, were learned during game-play (Bryant and Miikkulainen 2007).

3.2.7 Evolutionary optimisation of topology and connection weights for neural network-based game agent controller representations

Although the use of EA techniques within academic digital game research literature has primarily focused on the optimisation of connection weights for network-based game agent controllers, research has also been conducted into the optimisation of both the topology and connection weights for network-based game agent controllers. In the work of Spronck et al. (2003a) an EA with real-valued chromosome representation was used to generate the structure and connection weights for both multi-layered feed-forward and recurrent network-based game agent controllers for use in the strategy game *Picoverse*. Utilising a feature vector for the networks consisting of values representing the attack strength, defence strength and speed of both the game agent and an opponent agent, together with the direction of the opponent agent and a random value, a turn-based duel within *Picoverse* was used as the test environment for offline evolution against a single opponent agent that made use of a deterministic scripted strategy. Fitness was measured as the result of a number of duels between an evolved game agent and the opponent agent. Results from initial experiments indicated that a two-layered feed-forward neural network controller obtained a higher performance than other evolved network controllers, and an increase in the number of neurons in the hidden layers did not necessarily correspond to an increase in fitness. Based upon the observed behaviours of both the game agent and the opponent agent during testing, it was also found that, through the use of an evolved game agent, shortcomings could be detected in the scripted behaviour of the opponent agent. In order to address this, a further set of experiments were conducted using the evolved controller in a series of duels against an opponent agent that employed an improved set of scripted strategies. The authors have reported that results from this experiment showed a considerable drop in performance for the evolved controller, indicating that the evolved ship was optimised against the original scripted strategy used by the opponent agent during the initial experiment, hence the evolved game agent controller was unable to generalise to strategies unused during the evolutionary process (Spronck et al. 2003a).

In research conducted by Stanley et al. (2005a,b), D'Silva et al. (2005) and Yong et al. (2006) evolutionary methods have also been utilised for the automated generation of a neural network-based game agent controller's topology and connection weights within the machine learning-based game entitled *Neuroevolving Robotic Operatives (NERO)*. In Stanley and Miikkulainen (2002) the NeuroEvolution of Augmenting Topologies (NEAT) technique was presented, which makes use of a flexible chromosome representation scheme, combined with historical innovation markers, to permit efficient and compatible crossover between chromosomes representing different network topologies, and speciation techniques to protect structural innovations within evolving subpopulations of networks. The resulting technique (NEAT) allows for the increasing *complexification* of a neural network's structure appropriate to the complexity of the learning task, starting from a minimal network structure, over a number of generations of the evolutionary process (Stanley and Miikkulainen 2002). Extending the NEAT technique for use within real-time applications, Stanley et al. (2005a,b) developed the Real-Time Neuroevolution of Augmenting Topologies (rtNEAT) method that allows for the modification of a population of evolving agents in real-time, thus permitting a user to interact and influence the evolving populations. Rather than replacing an entire population of game agents during each generation of evolution, a single, potentially proficient, game agent is generated through the recombination of two highly fit parent chromosomes and used to replace one of the worst performing individuals in the current population. By periodically performing the generation of potentially fit individuals and replacement of poorly performing individuals according to a set number of simulation steps, game agent behaviours

may appear to evolve in a continuous manner, rather than intermittently, as would be normally the case when replacement occurs for an entire population on a generational basis. The combat training game *NERO* has been designed and built around the concept of explicit player controlled reinforcement learning, implemented using the rtNEAT method, and has been shown to be capable of efficiently and robustly evolving neural network controlled game agents in real-time (Stanley et al. 2005a,b).

In D'Silva et al. (2005), further research was performed using the rtNEAT method with *NERO* in order to overcome the problem of preserving previously generated useful game agent behaviours while new, unrelated behaviours are being evolved. A technique known as *milestoning* was introduced in which a pool of chromosomes from previously evolved game agents is maintained and individuals from this pool are periodically inserted into the evolving population. By retaining highly fit individuals in the milestone pool throughout the course of the evolutionary process, such prior knowledge may be periodically incorporated into the evolving population by randomly selecting an individual from the pool for recombination with a highly fit individual from the current population. When tested within the *NERO* game, results indicated that the use of milestoning is effective at retaining learned knowledge within game agents evolved in real-time, thus allowing game agents to be evolved to successfully learn unrelated behaviours (D'Silva et al. 2005).

Research conducted by Yong et al. (2006) further advanced the real-time generation of game agents for *NERO*, based on the use of the rtNEAT method, and introduced an approach for incorporating player generated advice into the evolutionary process. When evolving game agent behaviours for a difficult learning task, potentially prolonged periods of exploration during learning may occur. If the required behaviours are readily apparent to the player, interactions from the player may be harnessed to effectively guide the evolutionary process. In a game, such as *NERO*, where a player guides game agent development through a training regime using a control interface, it may be easier to specify desired game agent behaviours using natural language methods. By encoding such desired behaviours as advice structures, typically containing a set of IF-THEN rules, neural network representations of the advice may be generated and subsequently spliced into the neural network representation of the currently evolving game agent within the evolving population. It was reported that such advice networks may be further refined by the evolutionary process as part of the overall representation of a game agent, thus leading to the more efficient generation of task-specific game agent behaviours. When testing the use of player-generated advice within *NERO*, results indicated that the use of pertinent advice improves both the efficiency of the learning algorithm and the performance of the game agent with regard to learning task-specific behaviours. Subsequently, when a conflicting learning task was specified, the authors found that the original advice had been successfully refined, during further evolution, to enable the game agent to perform the new task, implying that the advice structures gradually became integrated into the evolved game agent representation (Yong et al. 2006).

The NEAT algorithm has also been extended in the work of Whiteson et al. (2005) in order to address the issue of feature selection by automatically determining an optimal set of inputs for the evolved networks during the evolutionary process. Incorporating the feature selection problem into the learning task, the proposed Feature Selective NeuroEvolution of Augmenting Topologies (FS-NEAT) method avoids the need for the selection of input features or explicit feature labelling by a domain expert, and the use of a separate learning algorithm to determine a useful set of features using meta-learning. In NEAT the evolutionary process begins with an initial population of simple networks containing a number of inputs directly connected to the network outputs, which makes the assumption that the choice of inputs, selected by a domain expert, are relevant to the learning task. By contrast, in

FS-NEAT the initial population of networks contains a single connection between a randomly selected input and output, with subsequent inputs being connected as the complexification of the network continues using the standard NEAT technique. In order to evaluate FS-NEAT a number of experiments were conducted which compared a game agent controller for a 2D racing simulation (*Robot Auto Racing Simulator*) generated using FS-NEAT against a game agent controller generated using NEAT. Based on the average score obtained by each controller, tested over a number of runs on a single track, the results showed that FS-NEAT produced controllers which outperformed those generated using NEAT. Subsequently, the FS-NEAT game agent controllers were evolved faster, used substantially less inputs, and contained a more compact network topology than the controllers generated using NEAT. When experiments were conducted using varying sizes of input feature sets, the authors found that NEAT always made use of all available inputs, with a resulting degradation in game agent performance as the size of the available feature set increased. Conversely, proficient controllers generated using FS-NEAT were found to make use of an almost constant subset of the available input features and showed consistent performance, regardless of the increasing availability of input features for selection (Whiteson et al. 2005).

3.2.8 Alternative approaches to hybrid evolutionary techniques

Although there is a predilection, within the academic digital game research literature, for combining EA techniques with neural networks, alternative hybridisations have also been reported in the literature. One such alternative hybrid technique has been proposed in Louis and McDonnell (2004), Miles et al. (2004a,b) and Louis and Miles (2005). In this research combinations of a GA and a Case-Base Reasoning (CBR) approach have been utilised to generate game agent strategies within a resource allocation-based strategy simulation entitled *Strike Ops*. In the Case Injected Genetic Algorithm (CIGAR) system, a CBR approach has been used, whereby a case-base of previously successful solutions to a learning task is maintained and used to periodically *inject* similar, useful solutions into the evolving population of a GA, thereby combining the advantages of both CBR and GA paradigms. The case base is initially populated with successful cases recorded from hand-coded strategies and human player strategies obtained during previous games. These are further evolved offline to generate optimal winning strategies (Louis and McDonnell 2004; Miles et al. 2004a,b; Louis and Miles 2005). Each case contains the chromosome of a potential solution together with its fitness value and a generational marker. When a solution to a new learning task is required, a percentage of the initial population for the GA are drawn from the case base. New cases are added as the fitness value of the fittest solutions in the GA population increase. By making use of a Hamming distance-based similarity metric and a *probabilistic closest to the best* case injection strategy, periodically cases that are similar to the most-fit solution in the current population replace the least-fit solutions, thus biasing the genetic search toward areas that generated previously successful solutions. The research carried out reported that the case injected GA produced effective game agent strategies that were reactive to both computer generated and human game-play strategies (Louis and McDonnell 2004; Miles et al. 2004a,b; Louis and Miles 2005).

3.3 Reinforcement learning approaches in digital games

Reinforcement learning comprises a set of algorithms and techniques focused on maximising an accumulated discounted reward over a period of time in response to a series of actions

performed within an environment. Through exploration and exploitation of a state-action space, based on feedback from interactions with an environment, a control policy can be learned that maximises the reward for a sequence of actions without requiring explicit training from a domain expert (Sutton and Barto 1998). Reinforcement learning techniques, such as the Temporal Difference learning methods (Sutton 1988) Q-Leaning (Watkins and Dayan 1992) and Sarsa (Rummery and Niranjan 1994; Sutton 1996), have been applied within academic digital game research in order to learn control policies for both game agents and game agent strategies, using both standard (*flat*) and *hierarchical* reinforcement learning approaches. Although traditional reinforcement learning techniques may be considered as unsuitable for digital games, due to issues associated with the size of state-action spaces and the complexity of digital games (Miikkulainen et al. 2006), the nature of exploration-based learning through an agent's interactions with its environment, underpinned by the theory of sequential decision processes, would seem complementary to the creation of game AI. However, before such techniques may be widely used within digital games, research must overcome the limitations associated with manipulating and scaling to large, complex state-action spaces (Baekkelund 2006; Miikkulainen et al. 2006). The following subsection will be split into the two subsections corresponding to the use of flat and hierarchical reinforcement learning approaches, as reported in the literature.

3.3.1 Flat reinforcement learning approaches used within digital games

In research conducted by Duan et al. (2002), Q-Learning was used to determine a control policy for game agents within an N-Person Iterated Prisoners Dilemma test bed, entitled *Escape*, in which a game agent must cooperate or compete with opponent agents in order to escape from a maze within a pre-specified time limit. Using a flat reinforcement learning approach, an optimal control policy for the game agent was found using offline value iteration, making use of a database of game agent states recorded during each simulation step, in order to generate an optimal set of actions for the game agent. By performing an update to the control policy every time a game agent participated in an interaction with an opponent agent, the authors reported that cooperation between the game agent and opponent agents was found to improve, therefore leading to less predictable and more believable game agent behaviours during game-play (Duan et al. 2002).

Similarly, Graepel et al. (2004) have made use of a flat reinforcement learning approach for the generation of a control policy for game agents within the fighting game *Tao Feng: Fist of the Lotus*.[15] Investigating the use of both linear and function approximation value function representations, the Sarsa(λ) algorithm was used offline against deterministic FSM controlled opponent agents, in order to generate an optimal game agent control policy. Using a set of features relating to the game environment, game agent and opponent agent as states for the learning algorithm, initial investigations were performed using a linear value function representation and a reward function that promotes the use of aggressive actions by the game agent. Results showed that when a less explorative action selection policy was used the game agent selected defensive actions more frequently than aggressive actions. By contrast, when a more explorative action selection policy was used, the resulting control policy became more optimal, favouring aggressive actions, at the expense of an increase in the time taken to learn the policy due to the increased exploration used during action selection. When a reward function was used that promotes defensive game agent behaviours, the authors found that an optimal control policy was obtained which was able to exploit loopholes in the determin-

[15] Microsoft Game Studios.

istic opponent agent AI. Further experimentation using a neural network for value function approximation resulted in a control policy similar to the use of a linear value function representation with the aggressive reward function. Although the use of a neural network helped smooth the learning curve, less exploration occurred during action learning (Graepel et al. 2004).

A flat reinforcement learning approach was also used in the work of Galway et al. (2007), for game agent control within a variation of the arcade game *Pac-Man*. Investigating the use of both the Sarsa and Sarsa(λ) control algorithms, a game agent controller was successfully generated in real-time using a linear value function representation. Decomposing the game environment into a finite number of states, a predefined period of value iteration was performed prior to selecting an action for the game agent, with updates to the value function occurring over a number of episodes of learning, beginning with the state associated with the current location of the game agent. A greedy action selection policy was then used to select the action for the current state corresponding to the state-action pair that achieved the largest value during learning. Performing experiments to investigate the use of prior knowledge on the performance of the game agent, the authors found that incorporating knowledge about the game environment into both the reward values used and the choice of valid states made available to the learning algorithm helped improve the performance of the game agent on a learning task involving a single level of the game with multiple opponent agents that made use of a stochastic control strategy. The authors also reported that reinforcement learning was capable of generating appropriate game agent control in real-time within a dynamic digital game environment, however, both the task prioritisation and observed behaviour of the game agent were subject to the step-size and trace-decay algorithmic parameters used (Galway et al. 2007).

As part of their research into simulated RC car racing controllers (see Sect. 3.2.5 for further details), Lucas and Togelius (2007) utilised the Sarsa algorithm to generate action-value based controllers for single player point-to-point racing on a range of randomly generated tracks. Using both linear and function approximation methods for value function representation, with value function updates being performed after each simulation step, the authors found that, for all experiments performed, the use of reinforcement learning, when using a MLP network for value function representation, was unable to learn a suitable control policy. Similarly, the controllers generated using a table-based value function representation performed very poorly on the majority of the experiments conducted. When a (15+15) ES was used to optimise the values within both the table-based and network-based value function representations, the authors reported that results varied according to the learning task with evolved table-based representations producing marginally more proficient controllers. The Sarsa algorithm was also further applied to controllers that had already been evolved to perform sufficiently. Depending on the representation of the game environment used, it was found that controllers generated using the Sarsa algorithm showed an increase in performance when initialised with an evolved controller (Lucas and Togelius 2007).

In Bradley and Hayes (2005a,b) group utility functions were presented as a multi-agent reward mechanism and demonstrated in conjunction with the Sarsa(λ) algorithm for the generation of a game agent control policy within a simple foraging game, wherein multiple game agents cooperate or compete in order to repeatedly collect a number of randomly placed rewards. Extending the notion of a team utility function (Wolpert et al. 2001), in which teamwork is achieved during learning by rewarding all agents that belong to a group with a reward based on the sum of the rewards received by all the agents in the group. The group utility function removes the assumption that each agent is part of the same team with

the same goals. Instead, utility functions, other than the summation function, may be used to combine rewards from groups of agents, thus specifying different relationships between agents. Subsequently, hierarchies of utility functions can then be formed in order to provide control over the rewards specified to different groups of agents during reinforcement learning, which permits a variety of behaviours to be learned by different groups of agents. Using a high-level group utility function, specified as the negative standard deviation of the rewards received by all agents in a group, the reward received for a game agent, during experiments conducted using the Sarsa(λ) algorithm, is the summation of the individual reward received by the game agent plus a scaled reward determined by the group utility function, where the scaling factor reflects the game agent's relationship with the group. Experiments conducted into the generation of balanced game agent control policies within a foraging task made use of a deictic representation for both the game's state space and action space, coupled with a linear value function representation. In Bradley and Hayes (2005a), control policies generated for two game agents were compared when a negative standard deviation group utility function was used during learning for one game agent and a standard individual utility function was used during learning for the second game agent. Results indicated that use of the group utility function lead to a balanced performance by both game agents in comparison with the disparity in performance observed when using a standard reward function (Bradley and Hayes 2005a). Extending these experiments to the generation of control policies for competing teams of game agents, where a team contained between one and four game agents, Bradley and Hayes (2005b) found that the using a group utility function, a balanced performance was achieved by same sized teams of game agents. However, this came at the expense of the speed with which the teams of game agents completed the learning task, particularly in the smaller sized teams. It was concluded that in order to achieve a balance in the learned performance of game agent teams, a trade off must be made between the amount of balanced behaviour required and the speed with which task achieving behaviours are performed (Bradley and Hayes 2005b).

3.3.2 Hierarchical reinforcement learning approaches used within digital games

Limitations regarding the use of a flat reinforcement learning approach, with linear value function representation, have been discussed in Maderia et al. (2004, 2006), where the Sarsa(λ) algorithm is used for the generation of a policy for game agent strategies within the real-time strategy game *Battleground*.[16] Using an hierarchical approach to strategic decision making, Maderia et al. (2004) used the Sarsa(λ) algorithm in order to learn high-level strategies based on an abstracted hierarchy of state representations, determined by terrain analysis performed on the game map. By using offline self-play against a rule-based opponent agent, reinforcement learning was used to learn high-level strategic decisions, which were then passed to rule-based game agents in order to perform low-level behaviours. Thus, the learning task was constrained to a subset of the overall decision-making system. Experiments conducted by playing the Sarsa(λ) controlled game agent against both a rule-based opponent agent and a random opponent agent resulted in game agent performances that largely outperformed the random opponent and which were very close in performance to the rule-based game agent. However, the authors suggested that the need for a simplified state representation, due to the use of a look-up table for value function representation, was a limitation in the strategies that were produced. Though, this may potentially be overcome using a more detailed state representation in conjunction with a function approximation method

[16] TalonSoft.

for value function representation (Maderia et al. 2004). Continuing the research into the use of the Sarsa(λ) algorithm with a neural network-based value function representation, Maderia et al. (2006) implemented a multi-layer network, trained using backpropagation, within the previous reinforcement learning architecture. Investigations using two network architectures were conducted; the first using a single network to provide a mapping between the abstracted state representation and an abstracted action space, and the second containing 45 networks to provide mappings from the abstracted state representation and individual actions. Learning was performed using self-play against a hand-coded rule-based opponent agent and results from evaluation games showed the performances of the game agent, using either of the network approaches to value function representation, were comparable with an average human player. As such, the authors have reported that the results indicated reinforcement learning is a suitable approach to game agent control within large-scale strategy games provided it is combined with a suitable function approximation method for value function representation. This concurs with results from the author's previous work (Maderia et al. 2006).

A hierarchical approach to reinforcement learning has also been used in the research conducted by Pfeiffer (2004) in order to learn control policies for game agents for a digital game version of the strategy game *Settlers of Catan*.[17] Decomposing high-level strategies into sets of low-level behavioural policies, the Sarsa algorithm was used with model-trees for value function representation in order to learn individual behavioural policies. Each behaviour policy determined a set of possible actions, and each action was represented by a model-tree. In order to overcome the need for a large number of training iterations, normally associated with the use of neural network-based function approximation, model-trees were used and trained using a series of random games before playing a further series of games in order to learn the behaviour policies. Four different approaches to hierarchical reinforcement learning were used during experiments conducted against random opponents, opponents using previously learned strategies and human opponents: (a) a *feudal* approach, which forces the Sarsa algorithm to learn both high-level strategies and behaviour policies simultaneously by allowing a high-level strategy to select a new behavioural policy during each simulation step, (b) a *module-based* approach, which permits a new high-level strategy only when the previous strategy's behaviour policy has been fully performed, (c) a *heuristic* approach, which makes used of hand-coded high-level strategies and attempts to learn the corresponding behaviour policies, and (d) a *guided* approach, which uses a hand-coded high-level strategy as a training example in order to further learn high-level strategies. Results from evaluation games, trained using the four approaches indicated that the heuristic approach generated game agent control policies that were capable of beating a human opponent. However, the author acknowledged the significance that the use of hand-coded high-level strategies brought to the overall control policy, independent of the learned low-level behaviour policies. Regardless, the author found that pre-training the model trees with domain knowledge improved the speed with which the game agent control policy was learned, leading to an overall improvement in the behaviours exhibited by the game agent (Pfeiffer 2004).

Both flat and hierarchical reinforcement learning approaches to the generation of game agent control policies have been investigated in the research conducted by Ponsen et al. (2006) within a real-time strategy game entitled *Battle of Survival*. In order to overcome the highly complex state-space involved, preventing the use of a linear value function representation, a deictic state representation was used in which game environment features were specified relative to the game agent. By further decomposing the learning task into a hierarchy of

[17] Franckh-Kosmos Verlags-GmbH & Co.

subtasks, a modified version of the Hierarchical Semi-Markov Q-Learning (HSMQ) algorithm (Dietterich 2000) has been implemented which incorporates a pseudo-reward function used during state-action value updates for non-primitive subtasks. The performance of game agent control policies generated using a deictic state representation with both the Q-Learning algorithm and the modified HSMQ algorithm were evaluated and compared for a navigation task featuring a single game agent and single opponent agent. After a number of episodes of learning, using randomly initialised versions of the learning task, evaluation was performed, with results showing a large improvement in performance for the game agent using the control policy generated with the modified HSMQ algorithm over the game agent that made use of the control policy generated with the Q-Learning algorithm. The authors also observed faster convergence to a sub-optimal control policy when using the hierarchical reinforcement learning approach (Ponsen et al. 2006).

3.3.3 Approaches inspired by reinforcement learning

Reinforcement learning has provided an inspiration to the development of a number of approaches for the generation of policies for both game agent control (Stanley et al. 2005a,b; Szita and Lõrincz 2006, 2007) and game agent strategies (Spronck et al. 2003b, 2006), however, the algorithms used have been based on the concept of reinforcement learning, rather than the explicit use of traditional reinforcement learning algorithms. For example, in the work of Szita and Lõrincz (2006, 2007) game agent control policies have been successfully generated for the digital games *Tetris*[18] (Szita and Lõrincz 2006) and *Ms. Pac-Man* (Szita and Lõrincz 2007). In Szita and Lõrincz (2006) value functions, represented by linear combinations of basis function, were iteratively learned by applying a modified version of the Cross-Entropy Method (CEM) optimisation algorithm (DeBoer et al. 2004) in order to determine the weights of the basis functions. Similarly, in Szita and Lõrincz (2007) CEM has been used to determine a control policy, comprising a prioritised rule-base learned from sets of pre-defined rules associated with observations and corresponding actions for both the game agent and the game environment (Szita and Lõrincz 2007). Reinforcement learning has also inspired the development of Dynamic Scripting by Spronck et al. (2003b, 2006) for the online creation and adaptation of rule-based policies, using a self-play mechanism, within the role-playing game *Neverwinter Nights*.[19] Likewise, the concept of reinforcement learning through explicit player guided supervision, rather than self-play, was used in the development of the combat training game *NERO* in research conducted by Stanley et al. (2005a,b), though evolutionary neural networks were used to implement the real-time reinforcement learning mechanism (please refer to Sect. 3.2.7 for further discussion on this implementation).

4 Summary & conclusion

In contrast to the AI used within commercial games, which typically make use of traditional AI techniques, such as the A* algorithm, FSM and planning algorithms (Buckland 2005; Hoang et al. 2005; Orkin 2005; Gorniak and Davis 2007), our review of the literature shows that academic digital game research has focused primarily on statistical machine learning techniques. Neural networks are used quite widely and indeed the application of neural net-

[18] The Tetris Company, LLC.

[19] BioWare Corp.

works to learning within digital game environments has shown a degree of success for a variety of learning tasks. In particular, the use of neural networks appears to be well suited when applied to player modelling, though such an approach requires a comprehensive dataset for training and validation. Traditional neural network training algorithms have been shown to be unsuitable for real-time use and are restricted to use as an offline learning mechanism. Alternative network implementation strategies, such as the use of an ensemble of networks and modular-based networks, potentially provide a more promising and suitable approach than the use of single network implementations. Through our review of the literature it has also become apparent that the hybridisation of neural networks and evolutionary methods feature predominantly within academic digital game research. As evolutionary techniques are a popular optimisation method, it is indeed no surprise that they have been coupled with neural networks for the generation of game agent controllers as such hybridisation enjoys the benefits of both paradigms. Though a number of successes have been achieved with the use of evolutionary and hybrid approaches, the stochastic nature of digital games tends to lead to noisy fitness evaluation, thus requiring greater numbers of games to be played during evaluation of a potential game agent controller. Approaches that have made use of pre-initialised populations have been shown to produce more proficient and robust game agent controllers. Similarly, the use of pre-initialised controller representations allow for the implementation of more efficient learning, thus providing a foundation for the implementation of online learning techniques. Successes have also been achieved with the use of reinforcement learning, though a suitable abstraction of a game environment's state-action space, coupled with the use of a flexible value function representation, are required for the use of reinforcement learning within more complex digital games. In the Appendix, Table 1 presents a summary of the key machine learning techniques we have reviewed, alongside the game agent representation upon which learning was performed, together with the corresponding game environment or test bed and associated reference. As can be seen in Table 1, considerable efforts have been made within academic digital game research to incorporate standard learning algorithms and both optimised and hybrid versions of learning algorithms into game AI for a wide range of game environments, from custom test bed environments to modified versions of commercial games.

Although neural networks have been predominantly used for game agent representation, they provide efficient operation when trained on sufficient data. However, training neural networks with traditional learning algorithms has proven to be inefficient and requires the generation of datasets containing examples to cover all possible situations that may be encountered by the game agent in order to obtain fully robust game agent behaviours (MacNamee and Cunningham 2003; Spronck et al. 2003a; Yannakakis et al. 2003; Chaperot and Fyfe 2006; Togelius et al. 2007b). Similarly, if neural networks are used for offline learning, especially when a player-modelling approach is being utilised, it is necessary to obtain datasets from a range of players to ensure the generation of useful game agent behaviours (McGlinchey 2003; Geisler 2004). It is apparent that the use of neural networks, particularly for online learning, requires an alternative training mechanism, such as the use of EA techniques. Subsequently, the hybridisation of such techniques has been shown to be more efficient and produces more proficient and adaptable game agents than the use of either technique alone (Yannakakis et al. 2003; Chaperot and Fyfe 2006). Even though evolutionary techniques may provide robust optimisation methods that do not require domain knowledge for effective use, canonical implementations typically suffer from slow convergence rates, with a number of difficulties encountered in the selection of a suitable chromosome representation scheme and fitness function for the learning task to be optimised (Baekkelund 2006). Similarly, noisy fitness evaluation must be countered,

along with the choice of a suitable replacement policy, in order to prevent game agent behaviours from appearing to change intermittently (Stanley et al. 2005a,b; Togelius and Lucas 2005; Parker and Parker 2007b). In the research presented, it has also been shown that seeding initial populations for evolution with pre-generated, proficient game agent controllers may help improve both the learning performance and the performance of the evolved game agents (Chaperot and Fyfe 2006; Togelius and Lucas 2006; Togelius et al. 2007a).

A number of issues also exist with the use of reinforcement learning techniques within digital game environments, including the need for a suitable state-action space abstraction and value function representation (Maderia et al. 2004, 2006; Ponsen et al. 2006). Slow convergence rates, the possibility of erratic behaviour during exploration and convergence to undesired behaviour, if the frequency and magnitude of feedback are not correctly balanced, also contribute to the difficulties associated with the use of reinforcement learning (Bradley and Hayes 2005a,b; Baekkelund 2006). However, it has been shown that the use of both hierarchical approaches and the incorporation of prior knowledge can help improve the performance of game agent controllers generated using reinforcement learning (Pfeiffer 2004; Ponsen et al. 2006; Galway et al. 2007). In a similar manner, using a pre-trained game agent controller as a basis for further learning has been shown to improve the performance of the resulting game agent (Lucas and Togelius 2007).

In conclusion, the use of digital games as an application domain for academic machine learning research has generated innovative approaches to the use of machine learning algorithms. At the same time, the limitations of various machine learning methods, when applied to the learning tasks encapsulated within such dynamic environments, have been shown. Subsequently, a more thorough understanding of the nature of digital game environments and the problems associated with learning game agent control has been realised. Regardless of the learning algorithm chosen, or genre of digital game used, the application of machine learning to digital games has given rise to a number of common issues. These include the choice of a suitable and flexible game agent representation scheme and feature vector, the necessity for contextually appropriate and visibly intelligent game agent behaviours, and the need for constraint and guidance over the learning process. In order to make significant advances to the growing field of digital game research, the focus of machine learning research within this application domain should be concentrated on these issues, with the pursuit of online learning mechanisms as the overarching goal. Although this focus necessitates the development of solutions to overcome the issues imposed by both digital game environments and the use of learning within such environments, current academic digital game research has shown some promising and encouraging results regarding the feasibility of adopting machine learning techniques for use within digital games. Only once these issues are resolved will the adoption of machine learning techniques be fully embraced by commercial game developers. In the meantime, academic research should continue its attempts to overcome these issues, thus helping to reduce the existing disparity between academic digital game research and the AI techniques currently utilised by game developers.

Appendix: Machine learning techniques summary

See Table 1.

Table 1 Machine learning techniques used within academic digital game research

Learning technique	Game agent representation	Game environment	Reference
Backpropagation	Multi-layer perceptron	*Motocross the force*	Chaperot and Fyfe (2006)
	Multi-layer perceptron	Simulated racing	Togelius et al. (2007b)
	Multi-layer perceptron	Simulated social environment	MacNamee and Cunningham (2003)
	Multi-layer perceptron	*Soldier of fortune 2*	Geisler (2004)
	Multi-layer perceptron (ATA[a])	*Legion-I*	Bryant and Miikkulainen (2003)
	Multi-layer perceptron (ATA)	*Legion-II*	Bryant and Miikkulainen (2006a)
	Multi-layer perceptron (Ensemble)	*Quake II*	Bauckhage and Thurau (2004)
Backpropagation (LM[b])	Multi-layer perceptron	*FlatLand*	Yannakakis et al. (2003)
Backpropagation (bagging)	Multi-layer perceptron (ensemble)	*Motocross the force*	Chaperot and Fyfe (2006)
	Multi-layer perceptron (ensemble)	*Soldier of fortune 2*	Geisler (2004)
Backpropagation (boosting)	Multi-layer perceptron	*Motocross the force*	Chaperot and Fyfe (2006)
	Multi-layer perceptron (Ensemble)	*Soldier of fortune 2*	Geisler (2004)
SOM	Self-organising map	*Pong*	McGlinchey (2003)
SOM & Backpropagation (LM)	Self-organising map & multi-layer perceptron	*Quake II*	Thurau et al. (2003)
Evolutionary algorithm	Single-layer perceptron	*Cellz*	Lucas (2004)
	Multi-layer perceptron	Simulated racing	Togelius and Lucas (2005)
	Multi-layer perceptron	Simulated racing	Togelius and Lucas (2006)
	Rule-base	*Wargus*	Ponsen and Spronck (2004)
Genetic algorithm	Single-layer perceptron	*Xpilot*	Parker et al. (2005b)
	Multi-layer perceptron	*Dead end*	Yannakakis et al. (2004)
	Multi-layer perceptron	*FlatLand*	Yannakakis et al. (2003)

Table 1 continued

Learning technique	Game agent representation	Game environment	Reference
	Multi-layer perceptron	*Pac-Man*	Yannakakis and Hallam (2004)
	Multi-layer perceptron	*Motocross the force*	Chaperot and Fyfe (2006)
	Neural network (modular)	*Xpilot*	Parker and Parker (2006b)
	Program-based operators	Simulated racing	Agapitos et al. (2007b)
	Rule-base	action game	Demasi and Cruz (2003)
	Rule-base	*Xpilot*	Parker et al. (2005a)
	Rule-base	*Xpilot*	Parker and Parker (2007b)
Genetic algorithm (parallel, steady-state)	Influence map tree	*Lagoon*	Miles et al. (2007)
Genetic algorithm (cyclic)	Rule-base	*Xpilot*	Parker et al. (2006)
	Rule-base	*Xpilot*	Parker and Parker (2006a)
Genetic algorithm (queue)	Multi-layer perceptron	*Xpilot*	Parker and Parker (2007a)
Genetic algorithm (NSGA-II^c)	Program-based operators	simulated racing	Agapitos et al. (2007b)
Genetic algorithm (case-injected)	Case-base	*Strike ops*	Louis and McDonnell (2004)
	Case-base	*Strike ops*	Miles et al. (2004a,b)
	Case-base	*Strike ops*	Louis and Miles (2005)
Evolutionary strategies	Single-layer perceptron	*Ms. Pac-Man*	Lucas (2005)
	Single-layer perceptron	Simulated racing	Lucas and Togelius (2007)
	Multi-layer perceptron	*Ms. Pac-Man*	Lucas (2005)
	Multi-layer perceptron	*Pac-Man*	Gallagher and Ledwich (2007)
	Multi-layer perceptron	Simulated racing	Agapitos et al. (2007a)
	Multi-layer perceptron	Simulated racing	Lucas and Togelius (2007)
	Multi-layer perceptron	Simulated racing	Togelius et al. (2007a)
	Neural network (modular)	Simulated racing	Togelius et al. (2007a)
	Recurrent network	Simulated racing	Agapitos et al. (2007a)
	Recurrent network	Simulated racing	Togelius et al. (2007a)

Table 1 continued

Learning technique	Game agent representation	Game environment	Reference
	Program-based operators	Simulated racing	Agapitos et al. (2007a)
	Program-based operators	Simulated racing	Togelius et al. (2007a)
	Value function (linear)	Simulated racing	Lucas and Togelius (2007)
	Value function (function approximation)	Simulated racing	Lucas and Togelius (2007)
Genetic programming	Program-based operators	*Pac-Man*	Koza (1992)
rtNEAT[d]	Neural network	*NERO*	Stanley et al. (2005a,b)
	Neural network	*NERO*	D'Silva et al. (2005)
	Neural network	*NERO*	Yong et al. (2006)
FS-NEAT[e]	Neural network	*Robot auto racing simulation*	Whiteson et al. (2005)
ESP[f]	Multi-layer perceptron (ATA)	*Legion-I*	Bryant and Miikkulainen (2003)
	Multi-layer perceptron (ATA)	*Legion-II*	Bryant and Miikkulainen (2006b)
	Multi-layer perceptron (ATA)	*Legion-II*	Bryant and Miikkulainen (2007)
PBIL[g]	Finite state machine & Rule-base	*Pac-Man*	Gallagher and Ryan (2003)
Q-learning	Value function	*Escape*	Duan et al. (2002)
	Value function	*Battle of survival*	Ponsen et al. (2006)
Q-learning (HSMQ[h])	Value function	*Battle of survival*	Ponsen et al. (2006)
Sarsa	Value function (linear)	*Pac-Man*	Galway et al. (2007)
	Value function (linear)	Simulated racing	Lucas and Togelius (2007)
	Value function (model tree)	*Settlers of catan*	Pfeiffer (2004)
	Value function (function approximation)	Simulated racing	Lucas and Togelius (2007)
Sarsa(λ)	Value function (linear)	*Battleground*	Maderia et al. (2004)
	Value function (linear)	Foraging game	Bradley and Hayes (2005a,b)

Table 1 continued

Learning technique	Game agent representation	Game environment	Reference
	Value function (linear)	*Pac-Man*	Galway et al. (2007)
	Value function (linear)	*Tao Feng*	Graepel et al. (2004)
	Value function (function approximation)	*Battleground*	Maderia et al. (2006)
	Value function (function approximation)	*Tao Feng*	Graepel et al. (2004)

[a] Adaptive team of agents
[b] Levenberg–Marquardt
[c] Non-dominated sorting genetic algorithm II
[d] Neuroevolution of augmenting topologies
[e] Feature selection neuroevolution of augmenting topologies
[f] Enforced sub-populations
[g] Population-based incremental learning
[h] Hierarchical semi-Markov Q-learning

References

Agapitos A, Togelius J, Lucas SM (2007a) Evolving controllers for simulated car racing using object orientated genetic programming. In: Thierens D et al (eds) Proceedings of the 2007 genetic and evolutionary computation conference. ACM Press, pp 1543–1550

Agapitos A, Togelius J, Lucas SM (2007b) Multiobjective techniques for the use of state in genetic programming applied to simulated car racing. In: Proceedings of the 2007 IEEE congress on evolutionary computation. IEEE Press, pp 1562–1569

Baekkelund C (2006) A brief comparison of machine learning methods. In: Rabin S (ed) AI game programming wisdom 3, 1st edn. Charles River Media, Hingham

Baluja S (1994) Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning. Technical report CMU-CS-94-163, Carnegie Mellon University

Bauckhage C, Thurau C (2004) Towards a fair 'n square aimbot—using mixtures of experts to learn context aware weapon handling. In: El-Rhalibi A, Van Welden D (eds) Proceedings of the 5th international conference on intelligent games and simulation. Eurosis, pp 20–24

Beyer H-G (2001) The theory of evolutionary strategies. Springer, Berlin

Billings D, Davidson A, Schaeffer J, Szafron D (2002) The challenge of poker. Artif Intell 134(1–2):201–240

Blumberg B, Downie M, Ivanov Y, Berlin M, Johnson MP, Tomlinson B (2002) Integrated learning for interactive synthetic characters. In: Proceedings of the 29th annual conference on computer graphics and interactive techniques. ACM Press, pp 417–426

Bradley J, Hayes G (2005a) Adapting reinforcement learning for computer games: using group utility functions. In: Kendall G, Lucas SM (eds) Proceedings of the 2005 IEEE symposium on computational intelligence and games. IEEE Press, Piscataway

Bradley J, Hayes G (2005b) Group utility functions: learning equilibria between groups of agents in computer games by modifying the reinforcement signal. In: Proceedings of the 2005 IEEE congress on evolutionary computation. IEEE Press, pp 1914–1921

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Bryant BD, Miikkulainen R (2003) Neuroevolution for adaptive teams. In: Proceedings of the 2003 IEEE congress on evolutionary computation. IEEE Press, pp 2194–2201

Bryant BD, Miikkulainen R (2006a) Exploiting sensor symmetries in example-based training for intelligent agents. In: Louis SJ, Kendall G (eds) Proceedings of the 2006 IEEE symposium on computational intelligence and games. IEEE Press, pp 90–97

Bryant BD, Miikkulainen R (2006b) Evolving stochastic controller networks for intelligent game agents. In: Proceedings of the 2006 IEEE congress on evolutionary computation. IEEE Press, pp 1007–1014

Bryant BD, Miikkulainen R (2007) Acquiring visibly intelligent behavior with example-guided neuroevolution. In: Proceedings of the 22nd National conference on artificial intelligence. AAAI Press, Menlo park, pp 801–808

Buckland M (2005) Programming game AI by example. Wordware Publishing, Plano

Buro M (1997) The othello match of the year: Takeshi Murakami vs. Logistello. ICCA J 20(3):189–193

Campbell M, Hoane AJ, Hsu F-H (2002) Deep blue. Artif Intell 134(1–2):57–83

Chaperot B, Fyfe C (2006) Improving artificial intelligence in a motocross game. In: Louis SJ, Kendall G (eds) Proceedings of the 2006 IEEE symposium on computational intelligence and games. IEEE Press, pp 181–186

Charles D (2003) Enhancing gameplay: challenges for artificial intelligence in digital games. In: Proceedings of digital games research conference 2003. University of Utrecht, 4–6 November 2003

Charles D, McGlinchey S (2004) The past, present and future of artificial neural networks in digital games. In: Mehdi Q, Gough N, Natkin S, Al-Dabass D (eds) Proceedings of the 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton, pp 163–169

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

DeBoer P-T, Kroese DP, Mannor S, Rubinstein RY (2004) A tutorial on the cross-entropy method. Ann Oper Res 134(1):19–67

DeJong KA, Sarma J (1992) Generation gaps revisited. In: Whitley D (ed) Foundations of genetic algorithms 2. Morgan Kaufmann, San Mateo, pp 19–28

Demasi P, Cruz AJ (2003) Online coevolution for action games. Int J Intell Games Simul 2:80–88

Dietterich TG (2000) An overview of MAXQ hierarchical reinforcement learning. In: Choueiry BY, Walsh T (eds) Lecture notes in computer science, vol. 1864. Springer, Berlin, pp 26–44

D'Silva T, Janik R, Chrien M, Stanley KO, Miikkulainen R (2005) Retaining learned behavior during real-time neuroevolution. In: Young M, Laird J (eds) Proceedings of the 1st artificial intelligence and interactive digital entertainment conference. AAAI Press, Menlo Park, pp 39–44

Duan J, Gough NE, Mehdi QH (2002) Multi-agent reinforcement learning for computer game agents. In: Mehdi QH, Gough NE (eds) Proceedings of the 3rd international conference on intelligent games and simulation. The University of Wolverhampton, pp 104–109

Elman JL (1990) Finding structure in time. Cog Sci 14:179–211

Evans R (2001) The future of AI in games: a personal view. Game Dev 8:46–49

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning. Morgan Kaufmann, pp 148–156

Fürnkranz J (2001) Machine learning in games: a survey. In: Fürnkranz J, Kubat M (eds) Machines that learn to play games. Nova Science Publishers, Huntington, pp 11–59

Gallagher M, Ledwich M (2007) Evolving Pac-Man players: can we learn from raw input? In: Proceedings of the 2007 IEEE symposium on computational intelligence in games. IEEE Press, pp 282–287

Gallagher M, Ryan A (2003) Learning to play Pac-Man: an evolutionary, rule-based approach. In: Proceedings of the 2003 IEEE congress on evolutionary computation. IEEE Press, pp 2462–2469

Galway L, Charles D, Black M (2007) Temporal difference control within a dynamic environment. In: Roccetti M (ed) Proceedings of the 8th international conference on intelligent games and simulation. Eurosis, pp 42–47

Geisler B (2004) Integrated machine learning for behavior modeling in video games. In: Fu D, Henke S, Orkin J (eds) Challenges in game artificial intelligence: papers from the 2004 AAAI workshop. AAAI Press, Menlo Park, pp 54–62

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading

Gomez F, Miikkulainen R (1997) Incremental evolution of complex general behavior. Adapt Behav 5:317–342

Gorniak P, Davis I (2007) SquadSmart hierarchical planning and coordinated plan execution for squads of characters. In: Schaeffer J, Mateas M (eds) Proceedings of the 3rd artificial intelligence and interactive digital entertainment conference. AAAI Press, Menlo Park, pp 14–19

Graepel T, Herbrich R, Gold J (2004) Learning to fight. In: Mehdi Q, Gough N, Natkin S, Al-Dabass D (eds) Proceedings of 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton, pp 193–200

Grand S, Cliff D, Malhotra A (1997) Creatures: artificial life autonomous software agents for home entertainment. In: Proceedings of the 1st international conference on autonomous agents. ACM Press, pp 22–29

Hagan MT, Menhaj M (1994) Training feed-forward networks with the Marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993

Haykin S (1994) Neural networks: a comprehensive foundation. Prentice-Hall, Upper Saddle River

Hoang H, Lee-Urban S, Muñoz-Avila H (2005) Hierarchical plan representations for encoding strategic game AI. In: Young M, Laird J (eds) Proceedings of the 1st artificial intelligence and interactive digital entertainment conference. AAAI Press, Menlo Park, pp 63–68

Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. MIT Press, Cambridge

Hsu F-H (2002) Behind deep blue: building the computer that defeated the World Chess Champion. Princeton University Press, Princeton

Jacobs R, Jordan M, Nowlan S, Hinton G (1991) Adaptive mixture of local experts. Neural Comput 3(1):79–87

Kirby N (2004) Getting around the limits of machine learning. In: Rabin SAI game programming wisdom 2, 1st edn. Charles River Media, Hingham

Kohonen T (1982) Self-organized formation of topologically correct feature maps. Biol Cybern 43:59–69

Koza JP (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

Laird JE, Van Lent M (1999) Developing an artificial intelligence engine. In: Proceedings of the 1999 game developers conference, pp 577–588

Laird JE, Van Lent M (2001) Human level AI's killer application: interactive computer games. AI Mag 22:15–25

Louis SJ, McDonnell J (2004) Learning with case-injected genetic algorithms. IEEE Trans Evol Comput 8:316–327

Louis SJ, Miles C (2005) Playing to learn: case-injected genetic algorithms for learning to play computer games. IEEE Trans Evol Comput 9:669–681

Lucas SM (2004) Cellz: A simple dynamic game for testing evolutionary algorithms. In: Proceedings of the 2004 IEEE congress on evolutionary computation. IEEE Press, pp 1007–1014

Lucas SM (2005) Evolving a neural network location evaluator to play Ms. Pac-Man. In: Kendall G, Lucas SM (eds) Proceedings of the 2005 IEEE symposium on computational intelligence and games. IEEE Press, pp 203–210

Lucas SM, Kendall G (2006) Evolutionary computation and games. IEEE Comput Intell Mag. February, pp 10–18

Lucas SM, Togelius J (2007) Point-to-point car racing: an initial study of evolution versus temporal difference learning. In: Proceedings of the 2007 IEEE symposium on computational intelligence and games. IEEE Press, pp 260–267

MacNamee B, Cunningham P (2001) Proposal for an agent architecture for proactive persistent non player characters. In: Proceedings of the 12th Irish conference on artificial intelligence and cognitive science, pp 221–232

MacNamee B, Cunningham P (2003) Creating socially interactive non player characters: the μ-SIC system. Int J Intell Games Simul 2:28–35

Maderia C, Corruble V, Ramalho G, Ratitch B (2004) Bootstrapping the learning process for semi-automated design of a challenging game AI. In: Fu D, Henke S, Orkin J (eds) Challenges in game artificial intelligence: papers from the 2004 AAAI workshop. AAAI Press, Menlo Park, pp 72–76

Maderia C, Corruble V, Ramalho G (2006) Designing a reinforcement learning-based adaptive AI for large-scale strategy games. In: Laird J, Schaeffer J (eds) Proceedings of the 2nd artificial intelligence and interactive digital entertainment conference. AAAI Press, Menlo Park, pp 121–123

Maes P (1995) Artificial life meets entertainment: lifelike autonomous agents. Commun ACM 38:108–114

Manslow J (2002) Learning and adaptation. In: Rabin SAI game programming wisdom, 1st edn. Charles River Media, Hingham

McGlinchey S (2003) Learning of AI players from game observation data. In: Mehdi Q, Gough N, Natkin S (eds) Proceedings of the 4th international conference on intelligent games and simulation. Eurosis, pp 106–110

Miconi T, Channon A (2006) The N-strikes-out algorithm: a steady-state algorithm for coevolution. In: Proceedings of the 2006 IEEE congress on evolutionary computation. IEEE Press, pp 1639–1646

Miikkulainen R, Bryant BD, Cornelius R, Karpov IV, Stanley KO, Yong CH (2006) Computational intelligence in games. In: Yen GY, Fogel DB (eds) Computational intelligence: principles and practice. IEEE Computational Intelligence Society, Piscataway, pp 155–191

Miles C, Louis SJ (2006) Towards the co-evolution of influence map tree based strategy game players. In: Louis SJ, Kendall G (eds) Proceedings of the 2006 IEEE symposium on computational intelligence and games. IEEE Press, pp 75–82

Miles C, Louis SJ, Cole N, McDonnell J (2004a) Learning to play like a human: case injected genetic algorithms for strategic computer gaming. In: Proceedings of the 2004 IEEE congress on evolutionary computing. IEEE Press, pp 1441–1448

Miles C, Louis SJ, Drewes R (2004b) Trap avoidance in strategic computer game playing with case injected genetic algorithms. In: Proceedings of the 2004 genetic and evolutionary computation conference. ACM Press, pp 1365–1376

Miles C, Quiroz J, Leigh R, Louis SJ (2007) Co-evolving influence map tree based strategy game players. In: Proceedings of the 2007 IEEE symposium on computational intelligence and games. IEEE Press, pp 88–95

Orkin J (2005) Agent architecture considerations for real-time planning in games. In: Young M, Laird J (eds) Proceedings of the 1st artificial intelligence and interactive digital entertainment conference. AAAI Press, Menlo Park, pp 105–110

Parker G, Rawlins G (1996) Cyclic genetic algorithms for the locomotion of hexapod robots. In: Proceedings of the 2nd world automation congress, pp 617–622

Parker GB, Doherty TS, Parker M (2005a) Evolution and prioritization of survival strategies for a simulated robot in xpilot. In: Proceedings of the 2005 IEEE congress on evolutionary computing. IEEE Press, pp 2410–2415

Parker G, Parker M, Johnson S (2005b) Evolving autonomous agent control in the xpilot environment. In: Proceedings of the 2005 IEEE congress on evolutionary computation. IEEE Press, pp 2416–2421

Parker GB, Doherty TS, Parker M (2006) Generation of unconstrained looping programs for control of xpilot agents. In: Proceedings of the 2006 IEEE congress on evolutionary computation. IEEE Press, pp 1216–1223

Parker M, Parker GB (2006a) Learning control for xpilot agents in the core. In: Proceedings of the 2006 IEEE congress on evolutionary computation. IEEE Press, pp 800–807

Parker GB, Parker M (2006b) The incremental evolution of attack agents in xpilot. In: Proceedings of the 2006 IEEE congress on evolutionary computation. IEEE Press, pp 969–975

Parker M, Parker GB (2006c) Using a queue genetic algorithm to evolve xpilot control strategies on a distributed system. In: Proceedings of the 2006 IEEE congress on evolutionary computation. IEEE Press, pp 1202–1207
Parker M, Parker GB (2007a) The Evolution of multi-layer neural networks for the control of xpilot agents. In: Proceedings of the 2007 IEEE symposium on computational intelligence and games. IEEE Press, pp 232–237
Parker GB, Parker M (2007b) Evolving parameters for xpilot combat agents. In: Proceedings of the 2007 IEEE symposium on computational intelligence and games. IEEE Press, pp 238–243
Pfeiffer M (2004) Reinforcement learning of strategies for settlers of catan. In: Mehdi Q, Gough N, Natkin S, Al-Dabass D (eds) Proceedings of the 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton, pp 384–388
Ponsen M, Spronck P (2004) Improving adaptive game AI with evolutionary learning. In: Mehdi Q, Gough N, Natkin S, Al-Dabass D (eds) Proceedings of the 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton, pp 389–396
Ponsen M, Spronck P, Tuyls K (2006) Hierarchical reinforcement learning with deictic representation in a computer game. In: Schobbens P-Y, Vanhoof W, Schwanen G (eds) Proceedings of the 18th Belgium-Netherlands conference on artificial intelligence. University of Namur, pp 251–258
Pottinger D (2000) The Future of AI in games. Game Dev 8:36–39
Rumelhart DE, Hinton GE, Williams RJ (1989) Learning internal representations by error propagation. In: Rumelhart DE, McClelland JL (eds) Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. MIT Press, Cambridge
Rummery GA, Niranjan M (1994) On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, UK
Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3(3): 211–229
Samuel AL (1967) Some studies in machine learning using the game of checkers. ii – recent progress. IBM J Res Dev 11(6):601–617
Schaeffer J (1997) One jump ahead: challenging human supremacy in checkers. Springer, Berlin
Schaeffer J (2000) The games computers (and people) play. In: Zelkowitz M (ed) Advances in computers: 50. Academic Press, San Diego, pp 189–266
Schaeffer J, Vanden Herik HJ (2002) Games, computers, and artificial intelligence. Artif Intell 134:1–7
Schwab B (2004) AI game engine programming, 1st edn. Charles River Media, Hingham
Spronck P, Sprinkhuizen-Kuyper I, Postma E (2003a) Improving opponent intelligence through offline evolutionary learning. Int J Intell Games Sim 2:20–27
Spronck P, Sprinkhuizen-Kuyper I, Postma E (2003b) Online adaptation of game opponent AI in simulation and in practice. In: Mehdi Q, Gough N, Natkin S (eds) Proceedings of the 4th international conference on intelligent games and simulation. Eurosis, pp 93–100
Spronck P, Ponsen M, Sprinkhuizen-Kuyper I, Postma E (2006) Adaptive game AI with dynamic scripting. Mach Learn 63:217–248
Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evol Comput 10:99–127
Stanley KO, Bryant BD, Miikkulainen R (2005a) Real-time neuroevolution in the NERO video game. IEEE Trans Evol Comput 9:653–668
Stanley KO, Bryant BD, Miikkulainen R (2005b) Evolving neural network agents in the NERO video game. In: Kendall G, Lucas SM (eds) Proceedings of the 2005 IEEE symposium on computational intelligence and games. IEEE Press, Piscataway, NJ
Sutton RS (1988) Learning to predict by the method of temporal differences. Mach Learn 3:9–44
Sutton RS (1996) Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Touretzky DS, Mozer MC, Hasselmo ME (eds) Advances in neural information processing systems 8. MIT Press, Cambridge, pp 1038–1044
Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
Szita I, Lõrincz A (2006) Learning tetris using noisy cross-entropy method. Neural Comput 18:2936–2941
Szita I, Lõrincz A (2007) Learning to play using low-complexity rule-based policies: illustrations through Ms. Pac-Man. J Artif Intell Res 30:659–684
Tesauro G (1995) Temporal difference learning and td-gammon. Commun ACM 38(3):58–68
Thurau C, Bauckhage C, Sagerer G (2003) Combining self-organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In: Mehdi Q, Gough N, Natkin S (eds) Proceedings of the 4th international conference on intelligent games and simulation. Eurosis, pp 119–123
Togelius J, Lucas SM (2005) Evolving controllers for simulated car racing. In: Proceedings of the 2005 IEEE congress on evolutionary computation. IEEE Press, pp 1906–1913

Togelius J, Lucas SM (2006) Evolving robust and specialized car racing skills. In: Proceedings of the 2006 IEEE congress on evolutionary computation. IEEE Press, pp 1187–1194
Togelius J, Burrow P, Lucas SM (2007a) Multi-population competitive co-evolutions of car racing controllers. In: Proceedings of the 2007 IEEE congress on evolutionary computation. IEEE Press, pp 4043–4050
Togelius J, Nardi RD, Lucas SM (2007b) Towards automatic personalized content creation for racing games. In: Proceedings of the 2007 IEEE symposium on computational intelligence and games. IEEE Press, pp 252–259
Tozour P (2001) Influence mapping. In: DeLoura M (ed) Game programming gems, vol. 2. Charles River Media, Hingham, pp 287–297
Tozour P (2002) The evolution of game AI. In: Rabin S (ed) AI game programming wisdom, 1st edn. Charles River Media, Hingham
Watkins C, Dayan P (1992) Q-learning. Mach Learn 8:279–292
Whiteson S, Stone P, Stanley KO, Miikkulainen R, Kohl N (2005) Automatic feature selection in neuro-evolution. In: Proceedings of the 2005 genetic and evolutionary computation conference. ACM Press, pp 1225–1232
Wolpert D, Sill J, Tumer K (2001) Reinforcement learning in distributed domains: beyond team games. In: Proceedings of the 17th international joint conference on artificial intelligence. Morgan Kaufmann, pp 819–824
Woodcock S (1998) Game AI: the state of the industry. Game Dev 5:28–35
Woodcock S (1999) Game AI: The state of the industry. Game Dev 6:34–43
Woodcock S (2000) Game AI: the state of the industry. Game Dev 7:24–32
Woodcock S (2001) Game AI: the state of the industry 2000–2001. Game Dev 8:36–44
Woodcock S (2002) Game AI: the state of the industry 2001–2002. Game Dev 9:26–31
Yannakakis GN, Hallam J (2004) Evolving opponents for interesting interactive computer games. In: Schaal S, Ijspeert A, Billard A, Vijayakumar S, Hallam J, Meyer J-A (eds) From animals to animats 8: proceedings of the 8th international conference on simulation of adaptive behaviour. MIT Press, pp 499–508
Yannakakis GN, Levine J, Hallam J (2004) An evolutionary approach for interactive computer games. In: Proceedings of the 2004 IEEE congress on evolutionary computation. IEEE Press, pp 986–993
Yannakakis GN, Levine J, Hallam J, Papageorgiou M (2003) Performance, robustness and effort cost comparison of machine learning mechanisms in FlatLand. In: Proceedings of the 11th Mediterranean conference on control and automation. Rhodes, 18–20 June 2003
Yong CH, Stanley KO, Miikkulainen R, Karpov IV (2006) Incorporating advice into evolution of neural networks. In: Laird J, Schaeffer J (eds) Proceedings of the second artificial intelligence and interactive digital entertainment conference. AAAI Press, Menlo Park, pp 98–104