Neuroevolution in Games:
State of the Art and Open Challenges

Sebastian Risi and Julian Togelius

Abstract—This paper surveys research on applying neuroevolution (NE) to games. In neuroevolution, artificial neural networks are trained through evolutionary algorithms, taking inspiration from the way biological brains evolved. We analyse the application of NE in games along five different axes, which are the role NE is chosen to play in a game, the different types of neural networks used, the way these networks are evolved, how the fitness is determined and what type of input the network receives. The article also highlights important open research challenges in the field.

I. INTRODUCTION

The field of artificial and computational intelligence in games is now an established research field, but still growing and rapidly developing[1]. In this field, researchers study how to automate the playing, design, understanding or adaptation of games using a wide variety of methods drawn from computational intelligence (CI) and artificial intelligence (AI) [17, 76, 78]. One of the more common techniques, which is applicable to a wide range of problems within this research field, is neuroevolution (NE) [30, 147]. Neuroevolution refers to the generation of artificial neural networks (their connection weights and/or topology) using evolutionary algorithms. This technique has been used successfully for tasks as diverse as robot control [83], music generation [51], modelling biological phenomena [21] and chip resource allocation [43] among many others.

This paper surveys the use of neuroevolution in games (Figure 1). A main motivation for writing it is that neuroevolution is an important method that has seen continued popularity since its inception around two decades ago, and that there are numerous existing applications in games and even more potential applications. The researcher or practitioner seeking to apply neuroevolution in a game application could therefore use a guide to the state of the art. Another main motivation is that games are excellent testbeds for neuroevolution research (and other AI research) that have many advantages over existing testbeds, such as mobile robotics. This paper is therefore meant to also be useful for the neuroevolution researcher seeking to use games as a testbed.

SR is with the IT University of Copenhagen, Copenhagen, Denmark. JT is with the Department of Computer Science and Engineering, New York University, New York, USA. Emails: sebr@itu.dk, julian@togelius.com

[1]There are now two dedicated conferences (IEEE Conference on Computational Intelligence and Games (CIG) and AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)) as well as a dedicated journal (IEEE Transactions on Computational Intelligence and AI in Games). Furthermore, work in this field is published in a number of conferences and journals in neighbouring fields.
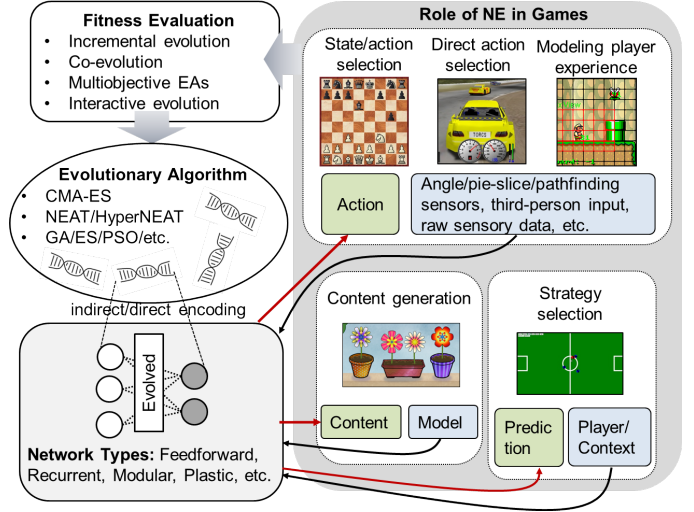

Fig. 1. **Neuroevolution in Games Overview**. An important distinction between NE approaches is the role that NE plays in a game, which is tightly coupled to the input the evolved neural network receives (e.g. angle sensors) and what type of output it produces (e.g. a request to turn). NE's role also directly influences the type of fitness evaluation. Different evolutionary algorithms support different network types and some methods can be more or less appropriate for different types of input representations.

A. Scope of this paper

In writing this paper, we have sought a broad and representative coverage of all kinds of neuroevolution to most kinds of games. While it is not possible to be exhaustive, we attempt to cover all of the main directions in this field and the most important papers. We only cover work where neuroevolution has in some way been applied to a game problem. By neuroevolution, we mean techniques where evolutionary computation or similar bio-inspired stochastic search/optimisation algorithms are applied to artificial neural networks (ANNs). We take an inclusive view of neural networks, including both weighted sums, self-organising maps and multi-layer perceptrons, but not e.g. expression trees evolved with genetic programming. By games, we refer to games that people commonly play as games. This includes non-digital games (e.g. board games and card games) and digital games (e.g. arcade games, racing games, strategy games) but not purely abstract games such as prisoner's dilemma, robotics tasks or non-game benchmarks for reinforcement learning, such as pole balancing. We acknowledge that there are bound to be some gray areas, as no delineation can be absolutely sharp.

There are several other surveys available that cover larger topics or topics that intersect with the topic of this paper. This

includes several surveys on CI/AI in games in general [77, 144], surveys on the use of evolutionary computation [72] or machine learning [36, 81], or surveys of approaches to particular research areas within the field such as pathfinding [4], player modelling [146] or procedural content generation [111].

B. Structure of this paper

The next section gives an overview of neuroevolution, the main idea behind it, and the main motivation for employing a NE approach in games. Section III details the first axes along which we analyse the role of neuroevolution in games, namely the type of role that the NE method is chosen to play. The different neural network types that are found in the NE games literature are reviewed in Section IV and Section V then explains how they can be evolved. The different ways fitness can be evaluated in games are detailed in Section VI, followed by a review of different ANN input representations in Section VII. Finally, Section VIII highlights important open challenges in the field.

II. NEUROEVOLUTION

The reasons for the wide applicability and lasting popularity of neuroevolution include that very many AI and control problems can be cast as optimization problems, where what is optimized is a general function approximator (such as a neural network). Another important reason for the attractiveness of this paradigm is that the method is grounded in biological metaphor and evolutionary theory. This section introduces the basic concepts and ideas behind neuroevolution. NE is motivated by the evolution of biological nervous systems and applies abstractions of natural evolution (i.e. *evolutionary algorithms*) to generate artificial neural networks. ANNs are generally represented as networks composed of interconnected nodes (i.e. *neurons*) that are able to compute values based on external inputs provided to the network. The "behavior" of an ANN is typically determined based on the architecture of the network and the strength (i.e. *weights*) of the connections between the neurons.

Training an ANN to solve a computational problem involves finding suitable network parameters such as its topology and/or weights of the synaptic connections. The basic idea behind neuroevolution is to train the network with an evolutionary algorithm, which is a class of stochastic, population-based search methods inspired by Darwinian evolution. An important design choice in NE is the genetic representation (i.e. genotype) of the neural network that the combination and mutation operators manipulate. For example, one of the earliest and most straightforward ways to encode an ANN with a fixed topology (i.e. the topology of the network is determined by the user) is based on the concatenation of the numerical network weight values into a vector of real numbers.

A. Basic Algorithm

The basic NE algorithm works as follows. A *population* of genotypes that encode ANNs is evolved to find a network (weight and/or topology) that can solve a computational problem. Typically each genotype is encoded into a neural network, which is then tested on a specific task for a certain amount of time. The performance or *fitness* of this network is then recorded and once the fitness values for the genotypes in the current population are determined, a new population is generated by slightly changing the ANN-encoding genotypes (mutation) or by combining multiple genotypes (cross-over). In general, genotypes with a higher fitness have a higher chance of being selected for reproduction and their offspring replaces genotypes with lower fitness values, thereby forming a new generation. This generational loop is typically repeated hundreds or thousands of times, in the hopes to finding better and better performing networks. For a more complete review of NE see Floreano et al. [30].

B. Why Neuroevolution?

For each of the tasks that are described in this paper, there are other methods that could potentially be used. Game strategies could be learned by algorithms from the temporal difference learning family, player models could be learned with support vector machines, game content could be represented as constraint models and created with answer set programming, and so on. However, there are a number of reasons why neuroevolution is a good general method to apply for many of these tasks and an interesting method to study in all cases. Figure 2 shows selected examples of NE in existing games that highlight some of its unique benefits.

1) Record-beating performance: For some problems, neuroevolution simply provides the best performance in competition with other learning methods ("performance" is of course defined differently for different problems). This goes for example for various versions of pole balancing, a classic reinforcement learning problem, where the CoSyNE neuroevolution method convincingly beat all other methods for most problem parameters [42]. In particular, it found solutions to the problem using fewer tries than any other algorithm. The best performance on the Keepaway Soccer problem, another popular reinforcement learning benchmark, is also exhibited by a neuroevolution method [137]. Neuroevolution can also perform very well on supervised learning tasks, as demonstrated by the performance of Evolino on several sequence prediction problems [104]. Here, a combination of neuroevolution and simple linear fitting could predict complex time-varying functions with lower error than any other method. In game domains, the winners of several game-based AI competitions are partly based on neuroevolution – see for example the winners of the recent 2K BotPrize [107, 108] and Simulated Car Racing Championship [9].

2) Broad applicability: Another benefit of NE is that it can be used for supervised, unsupervised and reinforcement learning tasks. Neuroevolution only requires some sort of numeric evaluation of the quality of its candidate networks. In this respect, neuroevolution is similar to other kinds of reinforcement learning (RL) algorithms, such as those from the temporal difference (TD) family [121]. However, if a dataset labelled with target values is provided NE can be used as a supervised learning algorithm similarly to how backpropagation is used.

Fig. 2. **Neuroevolution in Existing Games.** (a) NE is able to discover high-performing controllers for racing games such as TORCS [11]. (b) NE has also been successfully applied to commercial games, such as Creatures [44]. Additionally, NE enables new types of games such as GAR (c), in which players can interactively evolve particular weapons [46], or NERO (d), in which players are able to evolve a team of robots and battle them against other players [119].

3) Scalability: Compared to many other types of reinforcement learning, especially algorithms from the TD family, neuroevolution seems to handle large action/state spaces very well, especially when used for direct action selection [29, 48, 85, 86].

4) Diversity: Neuroevolution can draw on the rich family of diversity-preservation methods (such as niching) and multiobjective methods that have been developed within the evolutionary computation community. This enables neuroevolution methods to achieve several forms of diversity among its results, and so deliver sets of meaningfully different strategies, controllers, models and/or content [2, 136].

5) Open-ended learning: While NE can be used for RL in the same way that TD-learning can, one might argue that it could go beyond this relatively narrow formulation of reinforcement learning. In particular in cases where the topology of the network is evolved as well, neuroevolution could in principle support open-ended evolution, where behavior of arbitrary complexity and sophistication could emerge. Concretely, neuroevolution algorithms often search in a larger space than TD-based algorithms do.

6) Enables new kinds of games: New video games such as *Galactic Arms Race* (GAR; [46]), in which the player can interactively evolve particular weapons, NERO [119], which allows the player to evolve a team of robots and battle them against other players, or the *Petalz* video game [95, 97], in which the player can breed an unlimited variety of virtual flowers, would be difficult to realize with traditional learning methods. Evolutionary computation here provides unique affordances for game design, and some designs rely specifically on neuroevolution. In the case of games like Petalz and GAR, the games rely on the continuous complexification of the produced content, which is (naturally) supported and an integral part of certain NE methods. Additionally, in games like Petalz whose core game mechanic is the breeding of new flower types, employing evolutionary methods is a natural choice. An important example of a commercial game series that offers novel gameplay based on NE is *Creatures* [44]. In the first Creatures game, which was created in the mid-1990s, the player can breed and raise virtual pets called Norns, and teach them to survive in their environment. In contrast to most other games, the adaptive ANNs controlling the pets actually allow them to learn new behaviors guided by the player.

The fact that neuroevolution methods facilitate open-ended learning by incorporating a greater element of exploration, as highlighted in the previous section, also makes NE directly applicable to support new kinds of games.

C. Why not Neuroevolution?

While neuroevolution has multiple attractive properties, it is not a panacea and there are several reasons why other methods might be more suitable to particular problems. The most important one is that the evolved neural networks tend to have "black box" characteristics, meaning that a human cannot easily work out what they do by looking at them. This is a problem for game development and in particular quality assurance, as it becomes very hard to "debug" learned behavior. A problem with using neuroevolution online is that it is very hard to predict exactly what kind of behavior will be learned, something which can clash with the traditional design principles of commercial games.

III. ROLE OF NEUROEVOLUTION

The role that neuroevolution is chosen to play is the first of several axes along which NE's usage is analysed in this paper. According to our survey of the literature, in the vast majority of cases neuroevolution is used to learn to play a game or control an NPC in a game. The neural network can here have one of two roles: to *evaluate the value of states or actions*, so that some other algorithm can choose which action to take, or to *directly select actions* to take in a given state. But there are also other uses for neuroevolution. *Procedural content generation* (PCG) is an active area of research in game AI, and here evolvable neural networks can be used to *represent content*. Finally, neuroevolution can also *predict the experience or preferences* of players. Table I summarizes the usage of NE in a variety of different games.

This section will survey the use of neuroevolution in games in each role; each section will be ordered according to game genre.

A. State/action evaluation

The historically earliest and possibly most widespread use of neuroevolution in games is to estimate the value of board positions for agents playing classical board games. In these examples, a tree search algorithm of some kind (such as Minimax with alpha-beta pruning and/or other modifications) is used to search the space of future moves and counter-moves.

TABLE I
The Role of Neuroevolution in Selected Games. ES = evolutionary strategy, GA = genetic algorithm, MLP = multi-layer perceptron, MO = multiobjective, TP = third-person (input not tied to a specific frame of reference, e.g. number edible ghosts) , UD = user-defined network topology, PA = performance alone

NE Role (Section III)	Game	ANN Type (Section IV)	NE Methods (Section V)	Fitness Evaluation (Section VI)	Input Representation (Section VII)
State/action evaluation	**Checkers** [32]	MLP	UD, GA	Coevolution	TP (piece type)
	Chess [32]	MLP	UD, GA	PA (positional values)	TP (piece type)
	Othello [79]	MLP	Marker-based [34]	Cooperative coevolution	TP (piece type)
	Go (7×7) [38]	CPPN (MLP)	HyperNEAT	PA (score+board size)	TP (piece type)
	Ms. Pac-Man [71]	MLP	UD, ES	PA (average score)	Path-finding
	Simulated Car Racing [74]	MLP	UD, ES	PA (waypoints visited)	Speed, pos, waypoints
Direct action selection	**Quake II** [85, 86]	MLP	UD, GA	PA (kill count)	Visual Input (14×2)
	Unreal Tournament [135]	Recurrent, LSTM	UD, GA, NSGA-II	MO (damage&accuracy)	Pie-slice, way point, etc.
	Go (7×7) [118]	MLP	NEAT	Transfer Learning	Roving Eye (3×3)
	Simulated Car Racing [124]	MLP	UD, ES	Incremental Evolution	Rangefinders, waypoints
	Keepaway Soccer [122]	MLP	NEAT	Transfer Learning	Distances
	Battle Domain [105]	MLP	NEAT, NSGA-II	MO+Incremental	Angle, straight line
	NERO [119]	MLP	NEAT	Interactive Evolution	Rangefinders, pie-slice
	Ms. Pac-Man [106]	Modular MLP	NEAT, NSGA-II	MO (pills&ghosts eaten)	Path-finding
	Simulated Car Racing [29]	MLP	UD, GA	PA (distance)	Roving Eye (5×5)
	Atari [48]	CPPN (MLP)	HyperNEAT	PA (game score)	Raw input (16×21)
	Creatures [44]	Modular MLP	GA	Interactive Evolution	TP (e.g. type of object)
Selection between strategies	**Keepaway Soccer** [142, 143]	MLP	NEAT	PA (hold time)	Angle and distance
	EvoCommander [56]	MLP	NEAT	Interactive Evolution	Pie-slice, rangefinder
Modelling opponent strategy	**Texas Hold'em Poker** [66]	MLP	NEAT	PA (%hands won)	TP (e.g. size of pot, cost of a bet, etc.)
Content generation	**GAR** [46]	CPPN (MLP)	NEAT	Interactive Evolution	Model
	Petalz [97]	CPPN (MLP)	NEAT	Interactive Evolution	Model
Modelling player experience	**Super Mario Bros** [87]	MLP, Perceptron	UD, GA	PA (player preference)	TP (e.g. gap width, number deaths, etc.)

The role of the evolved neural network here is to evaluate the quality of a hypothetical future board state (the nodes of the search tree) and assign a numerical value to it. This quality value is supposed to be related to how close you are to winning the game, so that the rational course of action for a game-playing agent is to choose actions which would lead to board states with higher values. In most cases, the neural network is evolved using a fitness function based on the win rate of the game against some opponent(s), but there are also examples where the fitness function is based on something else, e.g. human playing style.

The most well-studied game in the history of AI is probably Chess. However, relatively few researchers have attempted to apply neuroevolution to this problem. Fogel et al. devised an architecture that was able to learn to play above master level solely by playing against itself [32].

There has also been plenty of work done on evolving board evaluators for the related, though somewhat simpler, board game Checkers. This was the focus of a popular science book by Fogel in 2001 [31], which chronicled several years' work on evolving Checkers players [16, 18]. The evolved player, called Blondie24, managed to learn to play at a human master level. However, interest in Checkers as a benchmark game waned after the game was solved in 2007 [101]. It has been shown that even very simple networks can be evolved to play Checkers well [54].

Another classic board game that has been used as a testbed for neuroevolution is Othello (also called Reversi). Moriarty and Miikkulainen used cooperative coevolution to evolve board evaluators for Othello [79, 80]. Chong et al. evolved board evaluators based on convolutional multi-layer neural

networks that captured aspects of board geometry [19]. In contrast, Lucas and Runarsson later used evolution strategies to learn position evaluators that were remarkably effective though they were simply weighted piece counters [73]. They also compared evolution to temporal difference learning for the same problem, a topic we will return to in Section VIII-B. Later work by the same authors showed that the n-tuple network, an evolvable neural network based on sampling combinations of board positions, could learn very good state evaluators [70]; n-tuple networks were also shown to perform better than other neural architectures on playing Checkers [3].

Go has in recent years become the focus of much research in AI game-playing. This is because this ancient Asian board game, despite having very simple rules, is very hard for computers to play proficiently. Until a few years ago, the best players barely reached intermediate human play level. The combination of Minimax-style tree search and board state evaluation that has worked so well for games such as Chess, Checkers and Othello largely breaks down when applied to Go, partly because Go has much higher branching factor than those games, and partly because it is very hard to calculate good estimates of the value of a Go board. Early attempts to evolve Go board state evaluators used fairly standard neural network architectures [69, 90]. It seems that such methods can do well on small-board Go (e.g. 5×5) but fail to scale up to larger board sizes [100]. Gauci et al. have attempted to address this by using HyperNEAT (see Section V), a neural network architecture specially designed to exploit geometric regularities to learn to play Go [38]; Schaul and Schmidhuber have tried to address the same issue using recurrent convolutional networks [103]. It should be noted that neuroevolution

is not currently competitive with the state of the art in this domain; the state of the art for Go is Monte Carlo Tree Search (MCTS), a statistical tree search technique which has finally allowed Go-playing AI to reach advanced human level [6].

But the use of neural networks as state value evaluators goes beyond board games. In fact any game can be played using a state value evaluation function, given that it is possible to predict which future states actions lead to, and that which there is a reasonably small number of actions. The principle is the same as for board games: search the tree of possible future actions and evaluate the resulting states, choosing the action that leads to the highest-valued state. This general method works even in the case of non-adversarial games, such as typical single-player arcade games, in which case we are building a Max-tree rather than a Minimax-tree. Further, sometimes good results can be achieved with very shallow searches, such as a one-ply search where only the resulting state after taking a single action is evaluated. A good example of this is Lucas' work on evolving state evaluators for Ms. Pac-Man [71] which has inspired a number of further studies [15, 106]. As Pac-Man never has more than four possible actions, this makes it possible to search more than one ply, as can be seen for example in the work of Borg Cardona et al. [15].

It is possible to evolve evaluators not only for states, but also for future actions. The neural network would in this case take the current state and potential future action as input, and return a value for that action. Control happens through enumerating over all actions, and choosing the one with the highest value. Lucas and Togelius compared a number of ways that neuroevolution could be used to control cars in a simple car racing game (with discrete action space), including evolving state evaluators and evolving action evaluators [74]. It was found that evolving state evaluators led to significantly higher performance than evolving action evaluators.

B. Direct action selection

It is not always possible to play a game (or control a game NPC) through evaluating potential future states or actions. For example, there might be too many actions from any given state to be effectively enumerable – a game like *Civilization* has an astronomical number of possible actions, and even a one-ply search would be computationally prohibitively expensive. (Though see Branavan et al.'s work on learning of action evaluators in *Civilization II* through a hybrid non-evolutionary method [5].) Further, there are cases where you do not even have a reliable method of predicting which state a future action would lead to, such as when learning a player for a game which you do not have the source code for, when the forward model for calculating this state would be too computationally expensive, or when the forward model is stochastic.

In such cases, neural networks can still play games and control NPCs through direct action selection. This means that the neural network receives a description of the current state (or some observation of the state) as an input and outputs which action to take. The output could take different forms, for example the neural network might have one output for each action, or continuous outputs that define some action space. In

many cases, the output dimensions of the network mirror or are similar to the buttons and sticks on a game controller that would be used by a human to play the game. (Note that the state evaluators and action evaluators discussed in the previous section also perform action selection, but in an *indirect* way.)

In a number of experiments with a simple car racing game, Togelius and Lucas evolved networks that drove the cars using a number of sensors as inputs, and outputs for steering and acceleration/braking [123, 124, 126]. It was shown that these networks could drive as least as well as a human player, and be incrementally evolved to proficiently drive on a wide variety of tracks. Using competitive coevolution, a variety of interestingly different playing styles could be generated. However, it was also shown that for a version of this problem, evolving networks to evaluate states gave superior driving performance compared to evolving networks to directly select actions [74]. This work on evolving car racing controllers spawned a series of competitions on simulated car racing; the first year's competition built on the same simple racing game [128], which was exchanged for the more sophisticated racing game *TORCS* for the remaining years [67, 68]. Several of the high-performing submissions were based on neuroevolution, usually in the role of action selector [11], but sometimes in more auxiliary roles [9].

Another genre where evolved neural networks have been used in the role of action selectors is first-person shooter (FPS) games. Here, the work of Parker and Bryant on evolving bots for the popular 90's FPS *Quake II* is a good example [85, 86]. In this case, the neural networks have different outputs for turning left/right, moving forward/backward and shooting. The FPS game which has been used most frequently for AI/CI experimentation is probably Unreal Tournament, as it has an accessible interface (called Pogamut [39]) and has been used in a popular competition, the *2K BotPrize* [50]. van Hoorn et al. evolved layered controllers consisting of several neural networks that implemented different components of good game-playing behavior: path following exploration and shooting [135]. The different networks output direct control signals corresponding to movement directions and shooting, and were able to override each other according to a fixed hierarchy. Schrum et al. evolved neural networks to behave in a human-like manner using the same testbed [107]. Another game with strong similarities to first-person shooter in terms of the capabilities of the agents (though a completely different focus for its gameplay) is the experimental game NERO, which sees each of its agents controlled by its own neural network evolved through a version of the popular NE method NEAT [119]. NEAT is explained in more detail in Section V.

In two-dimensional platform games, the number of actions possible to take at any given moment is typically very limited. This partly goes back to the iconic platform games, such as Nintendo's *Super Mario Bros*, being invented in a time when common game platforms had limited input possibilities. In Super Mario Bros, the action space is any combination of the buttons on the original Nintendo controller: up, down, left, right, A and B. Togelius et al. evolved neural networks to play the Infinite Mario Bros clone of Super Mario Bros using networks which simply had one output for each of the buttons

[130]. The objective, which met with only limited success, was simply to get as far as possible on a number of levels. A study by Ortega et al. subsequently used neuroevolution in the same role but with a different objective, evolving networks to mimic the behaviour of human players [84].

Whereas it might seem that when using neural networks for playing board games the networks should be used as state or action evaluators, there have been experiments with using neuroevolution in the direct action selection role for board games. This might mean having one network with an output for each board position, so that the action taken is the possible action which is associated with the highest-valued output; this is the case in the Checkers-playing network of Gauci et al. [37]. A more exotic variant is Stanley and Miikkulainen's "roving eye", which plays Go by self-directedly traversing the board and stopping where it thinks the next stone should be placed [118]. These attempts are generally not competitive performance-wise with architectures where the neural network evaluates states, and are done in order to test new network types or controller concepts.

Finally, there has been work on developing architectures for neural networks that can be evolved to play any of a large number of games. One example is Togelius and Schmidhuber's neural networks for learning to play any of a series of discrete 2D games from a space defined in a simple game description language [125] (this work later inspired the General Video Game Playing Competition; see Section VIII-D). More recently, Hausknecht et al. evolved neural networks of different types to play original Atari 2600 games using the Atari Learning Environment [47, 48]. The network outputs here were simply mapped to the Atari controller; performance varied greatly between different games, with evolved networks learning to play some games at master levels and many others barely at all.

C. Selection between strategies

For some games, it makes no sense to control the primitive actions of a player character, but rather to choose between one of several strategies to be played for a short time span (longer than a single time step). An example is the *Keepaway Soccer* task, which is a reinforcement learning benchmark with strongly game-like qualities as it is extracted from Robocup, a robot football tournament. Every time an agent has the ball, it selects one out of three static macro-actions, which is played out until the next ball possession [142, 143]. Whiteson et al. evolved neural networks for this task using NEAT with good results, but found that hybridising with other forms of reinforcement learning worked even better [142].

D. Modelling opponent strategy

In many cases, a player needs to be able to predict how its opponent will act in order to play well. Neuroevolution can be used in the specific role of predicting opponent strategy, as part of a player whose other parts might or might not be based on neuroevolution. Lockett and Miikkulainen evolved networks that could predict the other player's strategy in Texas Hold'em Poker, increasing the win rate of agents that used the model [66].

E. Content generation

Recently interest has increased in a field called procedural content generation (PCG) [111], in which parts of a game are created algorithmically rather than being hand-designed. Especially stochastic search methods, such as evolutionary algorithms, have shown promise in generating various types of content, from game levels to weapons and even the rules of the game itself; evolutionary approaches to PCG go by the name of *search-based* PCG [132].

In several applications, the content has been represented as a neural network. In these applications, the output characteristic (or decision surface) of the network is in some way interpreted as the shape, hull or other characteristic of the content artifact. The capacity for neural networks to encode smooth surfaces of essentially infinite resolution makes them very suitable for compactly encoding structures with continuous shapes. In particular, a type of neural network called a Compositional Pattern Producing Network (CPPN [115]) is of interest. CPPNs are a variation of ANNs that differ in the type of activation functions they contain and also the way they are applied. CPPNs are described in more detail in Section V. Examples of CPPNs used in content generation include the Galactic Arms Race video game (GAR [46]), in which the player can evolve CPPN-generated weapons, and the Petalz video game [95–97], in which the player can interactively evolve CPPN-encoded flowers. In another demonstration, Liapis et al. [64] evolved CPPNs to generate visually pleasing two-dimensional spaceships, which can be used in space shooter games. In more recent work, the authors augmented their system to allow the autonomous creation of spaceships based on an evolving interestingness criterion [65].

F. Modelling player experience

Many types of neural networks (including the standard multi-layer perceptron) are general function approximators, i.e. they can approximate any function to a given accuracy given a sufficiently large number of neurons. This is a reason that neural networks are very popular in supervised learning applications of various types. Most often, neural networks used for supervised learning are trained with some version of backpropagation; however, in some cases neuroevolution provides superior performance even for supervised learning tasks. This is in particular the case with preference learning, where the task is to learn to predict the ordering between instances in a data set.

One prominent use of preference learning, in particular preference learning through neuroevolution, is player experience modeling. Pedersen et al. trained neural networks to predict which level a player would prefer in a clone of Super Mario Bros [87]. The dataset was collected through letting hundreds of players play through pairs of different levels, recording the playing session of each player as well as the player's choice of which level in the pair was most challenging, frustrating and engaging. A standard multilayer perceptron was then given information on the player's playing style and features of both levels as inputs; the output was which of the two levels was preferred. Training these networks with neuroevolution, it was

found that player preference could be predicted with up to 91% accuracy. Shaker et al. later used these models to evolve personalised levels for this Super Mario Bros game, through simply searching the parameter space for levels that maximise particular aspects of player experience [110].

IV. NEURAL NETWORK TYPES

A variety of different neural network types can be found in the literature, and it is important to note that the type of ANN can significantly influence its learning abilities. The simplest neural networks are *feedforward*, which means that the information travels directly from the input nodes through the hidden nodes to the output nodes. In feedforward networks there are no synaptic connections that form loops or cycles. The simplest feedforward network is the perceptron, which is an ANN without any hidden nodes (only direct feed-forward connections between the inputs and the outputs), and where the activation function is either the simple step function or the identity function (in the latter case the perceptron is simply a weighted sum of its inputs) [98]. The classic perceptron without hidden nodes can only learn to correctly solve linearly separable problems, limiting its use for both pattern classification and control. The more complex multilayer perceptron (MLP) architectures with hidden nodes and typically sigmoid activation functions can also learn problems that are not linearly separable, and in fact approximate any function to a given accuracy given a suitable large number of hidden neurons [53]. While this capability is mostly theoretical, the greater representational capacity of MLPs over standard perceptrons can readily be seen when applying neuroevolution to NPC control in games. For example, Lucas [71] compared the performance of single and multi-layer perceptrons in a simplified version of the Ms. Pac-Man game and showed that the MLP reached a significantly higher performance.

In *recurrent networks*, the network can form loops and cycles, which allows information to also be propagated from later layers back to earlier layers. Because of these directed cycles the network can create and maintain an internal state that allows it to exhibit dynamic temporal properties and keep a memory of past events when applied to control problems. Simply put, recurrent networks can have memory, whereas static feedforward networks live in an eternal present. Being able to base your actions on the past as well as the present has obvious benefits for game-playing. In particular, the issue of sensory aliasing arises when the agent has imperfect information (true for any game in which not all of the game world is on screen at the same time), so that different states which are very different look the same to the agent [150]. Domains with this property are also called *non-Markovian* tasks, meaning that the state of the environment is not fully observable by the agent [114]. Recurrent and non-recurrent networks have been shown to perform very similarly in domains such as platform game playing [130], while recurrent networks have been shown to consistently outperform feedforward on a car racing task [136].

While most NE approaches in games focus on evolving monolithic neural networks, *modular* networks have recently shown superior performance in some domains [106]. A modular network is composed of a number of individual neural networks (modules), in which the modules are normally responsible for a specific sub-function of the overall system. For example, Schrum has recently shown that in the Pac-Man domain a modular network extension to NEAT performs better than the standard version which evolves monolithic (non-modular) networks. The likely explanation is that it is easier to evolve multimodal behavior with modular architectures [106]. That experiment featured modules which were initially undifferentiated but whose role was chosen by evolution. Explicitly functionally differentiated modules can also work well when the task lends itself to easy functional decomposition. van Hoorn et al. evolved separate neural modules for path-following, shooting and exploration; these modules were arranged in a hierarchical architecture, an arrangement which was shown to perform much better than a monolithic network [135]. Outside of games, there has been research on ways of encouraging neural networks to evolve modular structures [21], and also on evolving ensembles of neural networks [148] where the different networks have complementary fitness functions [1].

While most types of neural networks do not change their connection weights after initial training, *plastic* neural networks can change their connection weights at any time in response to the activation pattern of the network. This enables a form of longer-term learning than what is practically possible using recurrent networks. Plastic networks have been evolved to solve robot tasks in unpredictable environments which the controller does not initially know characteristics of [28], however they have not yet been applied to games to our best knowledge. It seems that plastic networks would be a promising avenue for learning controllers that can play multiple games or multiple versions of the same game. Another adaptive architecture that has shown promising results in learning from experience, is a *Long-Short Term Memory* (LSTM) network [40]. Instead of changing connection weights, LSTM networks can learn by remembering values for an arbitrary amount of time. We will return to the topic of life-time learning when we discuss important open challenges in Section VIII.

V. EVOLVING NEURAL NETWORKS

A large number of different evolutionary algorithms have been applied to NE, including genetic algorithms, evolution strategies and evolutionary programming [147]; in addition, other stochastic search/optimisation methods like particle swarm optimisation have also been used [26]. The method for evolving a neural network is tightly coupled to its genotypic representation. The earliest NE methods only evolved the weights of a network with a fixed user-defined topology and in the simplest *direct representation*, each connection is encoded by a single real value. This encoding, sometimes called *conventional neuroevolution* (CNE), allows the whole network to be described as a concatenation of these values, i.e. a vector of real numbers. For example, both Cardona et al. [15] and Lucas [71] directly evolved the weights of a MLP for Ms. Pac Man. Being able to represent a whole network as a vector of real numbers allows the use of any evolutionary algorithm which works on vectors of real numbers, including highly

efficient algorithms like the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [45, 55].

A significant drawback of the fixed topology approach is that the user has to choose the appropriate topology and number of hidden nodes a priori. Because the topology of a network can significantly affect its performance [55, 126, 129, 130], more sophisticated approaches evolve the network topology together with the weights. An example of such a method is NeuroEvolution of Augmenting Topologies (NEAT; [116]), which has shown that also evolving the topology together with the connection weights often outperforms approaches that evolve the weights alone. NEAT and similar methods also allow the evolution of recurrent networks that can solve difficult non-Markovian control problems (examples outside of games include pole balancing [42] and tying knots [75]). NEAT and its variants[2] have been applied successfully to a variety of different game domains, from controlling a simulated car in The Open Car Racing Simulator (TORCS) [10] or a team of robots in the NERO game [119] to playing Ms. Pac-Man [106] or Unreal Tournament [107]. Of particular interest in the context of this paper is an extension to NEAT that has been developed to increase its performance in strategic-decision making problems [61]. Kohl and Miikkulainen [61] demonstrate that by adding a new topological mutation that adds neurons with local fields to the network (RBF-NEAT) and a node mutation that biases the network architecture to cascade type of structures (Cascade-NEAT), the approach is able to more easily solve *fractured* strategic-decision task such as keepaway soccer. Fracture is defined by the authors as a "highly discontinuous mapping between states and optimal actions" within an ANN. It is likely that the more complex the game, the more it will have a fractured decision space.

Both the fixed-topology and topology-evolving approaches above use a direct encoding, meaning that each connection is encoded separately as a real number. Direct encoding approaches employ a one-to-one mapping between the parameter values of the network (e.g. the weights and neurons) and its genetic representation. One disadvantage of this approach is that parts of the solution that are similar must be discovered separately by evolution. Therefore, interest has increased in recent years in *indirect encodings*, which allow the reuse of information to encode the final network and thus very compact genetic representations (i.e. a high number of synaptic connections can be encoded by considerably fewer parameters in the corresponding genotype).

A promising indirect encoding that has been employed successfully in a variety of NE-based games are CPPNs [115] (see Section III-E). CPPNs are a variation of ANNs that differ in the type of activation functions they contain and also the way they are applied. While ANNs are traditionally employed as controllers, CPPNs often function as pattern-generators, and have shown promise in the context of procedurally generated content. Additionally, while ANNs typically only contain sigmoid activation functions, CPPNs can include these but also a variety of other activation functions like sine (to create

repeating patterns) or Gaussian (to create symmetric patterns). Importantly, because CPPNs are variations of ANNs they can also be evolved by the NEAT algorithm [120].

While CPPNs were initially designed to produce two-dimensional patterns (e.g. images), they have also been extended to evolve indirectly encoded ANNs. The main idea in this approach, called *HyperNEAT* [120] is to exploit geometric domain properties to compactly describe the connectivity pattern of a large-scale ANN. For example, in a board game like chess or checkers adjacency relationships or symmetries play an important role and understanding these regularities can enable an algorithm to learn general tactics instead of specific actions associated with a single board piece.

Hausknecht et al. [48] recently showed that HyperNEAT's ability to encode large scale ANNs enables it to directly learn Atari 2600 games from raw game screen data, outperforming human high scores in three games. This advance is exciting from a game perspective since it takes a step towards *general* game playing that is independent from a game specific input representation. We will return to this topic in Section VII.

Another form of indirect encodings are *developmental approaches*. In a promising demonstration of this approach, Khan and Miller [60] evolved a single developing and more complex neuron that was able to play checkers and beat a Minimax-based program. In contrast to the aforementioned indirect encodings like CPPNs, the neuron in Khan and Miller's work grows (i.e. develops) new synaptic connections while the game is being played. This growth process is based on an evolved genetic program that processes inputs from the board.

VI. FITNESS EVALUATION

The fitness of a board value estimator or ANN controlled NPC is traditionally determined based on its performance in a particular domain. This might translate to the score it achieves in a single player game or how many levels it manages to complete, or how many opponents it can beat or which ranking it can achieve in a competitive multiplayer game. Because many games are non-deterministic leading to *noisy fitness evaluation* (meaning that the same controller can score differently when evaluated several times on the same problem), evaluating the performance of a controller in multiple independent plays and averaging the resulting scores can be beneficial [100, 123]. In many cases it is possible to evolve good behaviour using a straightforward fitness function like the score of the game, which has the advantage that such fitness functions can typically not be "exploited" in the way composite fitness functions sometimes can (the only way to achieve a high score is to play the game well).

However, if the problem becomes too complex, it can be difficult to evolve the necessary behaviors directly. In such cases it can be helpful to learn them incrementally, by first starting with a simpler task and gradually making them more complex; this is usually called *staging* and/or *incremental evolution* [41]. For example, Togelius and Lucas [124] evolved controllers for simulated car racing with such an incremental evolution approach. In their setup, each car is first evaluated on one track and each time the population reaches a sufficiently high

---

[2]NEAT now also runs under the Unity Game Engine: https://github.com/lordjesus/UnityNEAT. A complete list of NEAT implementations can be found here: http://bit.ly/1J1II8O

fitness more challenging tracks are added to the evaluation and fitness averaged. The authors showed that by following such an incremental evolutionary approach, neurocontrollers with general driving skills evolve. No such general driving skills were observed when a controller was evaluated on all tracks simultaneously. Incremental evolution can also be combined with modularisation of neural networks so that each time a new fitness function is added, a new neural network module is added to the mix; this could allow evolution to solve problems where the acquisition of a new competence conflicts with an existing competence [135].

A form of training that can be viewed as a more radical version of incremental evolution is *transfer learning*. The goal of transfer learning is to accelerate the learning of a target task through knowledge gained during learning of a different but related source task. For example, Taylor et al. [122] showed that transfer learning can significantly speed up learning through NEAT from 3 vs. 2 to 4 vs. 3 robot soccer Keepaway. In a demonstration of applying transfer learning to two different but related games, Cardamone et al. [14] transferred a car racing controller evolved in TORCS to another open-source racing game called VDrift. By exploiting knowledge gained in TORCS, evolution was able to adapt the pre-existing model to the new game quicker than when the authors tried to evolve a controller in VDrift from scratch.

While incremental evolutionary approaches allow NE to solve more complex problems, designing a "good" fitness function can be a challenging problem, especially if the tasks become more and more complex. In particular, this is the case for competitive games where a good and reliable opponent AI is not available, and for games where an evolutionary algorithm easily finds strategies that exploit weaknesses in the game or in opponent strategies. A method to potentially circumvent such problems is *competitive coevolution* [15, 99, 117], in which the fitness of one AI controlled player depends on its performance when competing against another player drawn from the same population of from another population. The idea here is that the evolutionary process will supply itself with worthy opponents; in the beginning of an evolutionary run, players are tested against equally bad players, but as better players develop they will play against other players of similar skill. Thus, competitive coevolution can be said to perform a kind of automated incrementalisation of the problem; ideally, this would lead to *open-ended* evolution, where the sophistication and performance of evolved agents would continue increasing indefinitely due to an *arms race* [82, 99]. While open-ended evolution has never been achieved due to a number of phenomena such as cycling and loss of gradient, competitive coevolution can sometimes be very effective.

One of the earliest examples of coevolution in games was performed by Lubberts and Miikkulainen [69]. The authors coevolve networks for playing a simplified version of Go and show that coevolution can speed up the evolutionary search. These results have since been corroborated by other authors [100]. In a powerful demonstration of coevolution Blondie24 [31] reached expert-level by only playing against itself. In Khan and Miller [59] work, the authors showed that high performing ANNs evolve quicker through coevolution than through the evaluation of agents against a Minimax-based checkers player.

In another example, Cardona et al. [15] applied coevolutionary methods to coevolve the controllers for Ms. Pac Man and Ghost team controllers in Ms. Pac-Man. Interestingly, the authors discovered that it was significantly easier to coevolve controllers for Pac-man than for the team of ghosts, indicating that the success of a coevolutionary approach is very much dependent on the chosen domain and fitness transitivity (i.e. how much a solution's performance over one opponent correlates with its performance over other opponents).

In competitive coevolution, the opponents can be drawn from the same population, or from one or several other populations. This was investigated in a series of experiments in evolving car racing controllers, where it was found that using multiple populations improved results notably [126].

Coevolution does not have to be all about vanquishing one's opponents – there is also cooperative coevolution. Whereas in competitive coevolution fitness is negatively affected by the success of other individuals, in cooperative coevolution it is positively effected. Generally, an individual's fitness is defined by its performance in collaboration with other individuals. This can be implemented on a neuronal level, where every individual is a single neuron or synapse and its fitness is the average performance of the several networks it participates in. The CoSyNE neuroevolution algorithm, which is based on this idea, was shown to outperform all other methods for variants of the pole balancing problem, a classic reinforcement learning benchmark [42]. In another example, Cardamone et al. [12] demonstrated that CoSyNE is also able to create competitive racing controllers for the TORCS Endurance World Championship.

The conceptually closely related algorithms ESP and SANE have been used to evolve to strategies for Othello [80] and for the strategy game *Legion II* [7]. In another example, Whiteson et al. [140] showed that coevolution can successfully discover complex control tasks for soccer Keepaway players when provided with an adequate task decomposition. However, if the problem is made more complex and the correct task composition is not given, coevolution fails to discover high performing solutions.

For some problems, more than one fitness dimension need to be taken into account. This could be the case when there is no single performance criterion (for example an open-world game with no scoring or several different scores) or when there is a good performance indicator, but evolution is helped by taking other factors into account as well (for example, you want to encourage exploration of the environment). Further, "performance" might just be one of the criteria you are evolving your agent towards. You might be equally interested in human-likeness, believability or variety in agent behavior. There are several approaches that can be taken towards the existence of multiple fitness criteria. The simplest approach is just summing the different fitness functions into one; this has the drawback that it in practice leads to very unequal selection pressure in each of the fitness dimensions. Another approach is *cascading elitism*, where each generation contains separate selection events for each fitness function, ensuring

equal selection pressure [127].

A more principled solution to the existence of (or need for) multiple fitness functions is to use a *multiobjective evolutionary algorithm* (MOEA) [22, 24, 33]. Such algorithms try to satisfy all their given objectives (fitness functions), and in the cases where they partially conflict, map out the conflicts between the objectives. This map can then be used to find the solutions that make the most appealing tradeoffs between agents. These solutions, in which none of the objectives can be improved without reducing the performance of one of the other objectives, are said to be on the *Pareto Front*. Multiobjective evolution has been used to balance between pure performance and other objectives in several ways. For example, van Hoorn et al. [136] used an MOEA to balance between driving well and driving in a human-like manner (similar to recorded playtraces of human drivers) in a racing game. Agapitos et al. [2], working with a different racing game, used multiobjective evolution and several different characteristics of driving style to evolve sets of NPC drivers that had visibly different driving styles while all retaining high driving skill. Schrum and Miikkulainen [105] used the popular multi-objective evolutionary algorithm NSGA-II (Non-dominated Sorting Genetic Algorithm; [24]) to simply increase game-playing performance by rewarding the various components of good game-playing behavior separately.

Another promising NE approach, which might actually enable new kinds of games, is to allow a human player to interact with evolution by explicitly setting objectives during the evolutionary process, or even to act as a fitness function him or herself. Such approaches are known as *interactive evolution*. In the NERO video game [119], the player can train a team of NPCs for military combat by designing an evolutionary curriculum. This curriculum consists of a sequence of training exercises that can become increasingly difficult. For example, to train the team to perform general maze navigation the player can design increasingly complex wall configurations. Once the training of the agents is complete they can be loaded into battle and tested against teams trained by other players.

More recently, Karpov et al. [58] investigated three different ways of assisting neuroevolution through human input in OpenNERO (an open-source platform based on the original NERO). One approach was the *advice* method, in which users write short examples of code that are automatically converted into a partial ANN; these networks are then spliced into the evolving population. The other approach was *shaping*, in which users can influence the training by modifying the environment, similar to the setup in NERO [119]. The last approach was *demonstration*, in which users can control the agent manually and the recorded data is used to train the evolving networks in a supervised fashion. The authors showed that the three ways of human-assisted NE outperform unassisted NE in different tasks, further demonstrating the promise of combining NE with human expertise.

Another example of interactive evolution in games is Galactic Arms Race (GAR [46]), in which the players can discover unique particle system weapons that are evolved by NEAT. The fitness of a weapon in GAR is determined based on the numbers of times it was fired, allowing the players to implicitly drive evolution towards content they prefer. In Petalz [95, 97], a casual social game on Facebook, the core game mechanic is breeding different flowers. The player can interact with evolution by deciding which flowers to pollinate (mutation) or cross-pollinate (crossover) with other flowers.

Other interactive evolution approaches to game-like environments include the evolution of two-dimensional pictures on Picbreeder [109], three-dimensional forms [20], musical compositions [52], or even dance moves [25].

VII. INPUT REPRESENTATION

So far, we have discussed the role of neuroevolution in a game, types of neural networks, NE algorithms and fitness evaluation. Another interesting dimension by which to differentiate different uses of neuroevolution in games is what sort of input the network gets and how it is represented. Choosing the "right" representation can significantly influence the ability of an algorithm to learn autonomously. This is particularly important when the role of the neural network is to control an NPC in some way. For example, in a FPS game the world could be represented to the ANN as raw sensory data, as x and y coordinates of various in-game characters, as distances and angles to other players and objects, or as simulated sensor data. In Ms. Pac-Man the inputs could for example be the shortest distances to all ghosts or more high-level features such as the most likely path of the ghosts [8].

Additionally, which representation is appropriate depends to some extent on the type and on the role of the neural network in controlling the agent (see Section III). Some representations might be more appropriate for games with direct action selection, while other representations are preferable for a board or strategy game with state evaluation. In general, the type of input representation can significantly bias the evolutionary search and dictate which strategies and behaviors can ultimately be discovered.

In this section we cover a number of ways information about the game state can be conveyed to a neural network: as straight line sensors/pie slice sensors, pathfinding sensors, third-person data, and raw sensory data. Regardless of which representation is used, there are a number of important considerations. One of the most important is that all inputs need to be scaled to be within the same range, preferably within the range $[-1, 1]$. Another is that adding irrelevant inputs can be actively harmful to performance; while in principle neuroevolution should be capable of sorting useful from useless inputs on its own, drastic performance increases can sometimes be noted when removing information from the input vector [129].

A. Straight Line Sensors and Pie Slice Sensors

A good domain to exemplify the effect of different input representation is the simple car racing game studied by Togelius and Lucas [123, 124] and Togelius et al. [126]. Togelius and Lucas [123] investigated various sensor input representations and showed that the best results were achieved by using egocentric first-person information from rangefinder sensors, which return the distance to the nearest object in a given direction, instead of third-person information such

as the car's position in the track's frame of reference. The authors suggest that a mapping from third-person spatial information to appropriate first-person actions is likely very non-linear, thus making it harder for evolution to discover such a controller. (These results might also be explained by the underlying fractured decision space of such a mapping [61]; see Section V).

In addition to evolving appropriate control policies, NE approaches can also support automatic *feature selection*. Whiteson et al. [141] evolved ANNs in a car racing domain and showed that an extension to NEAT allows the algorithm to automatically determine an appropriate set of straight line sensors, eliminating redundant inputs.

Rangefinder sensors are also popular in other domains like FPS games. In NERO [119] each agent has rangefinder sensors to determine their distance from other objects and walls. In addition to rangefinder sensors, agents in NERO also have "pie slice sensors" for enemies. These are radar-like sensors which divide the 360 degrees around the agent in a predetermined number of slices. The ANN has inputs for each of the slices and the activation of the input is proportional to the distance of an enemy unit is within this slice. If multiple units are contained in one slice, their activations are summed. Similar sensors can be found in e.g. van Hoorn et al. [135]. While pie slice sensors are useful to detect discrete objects like enemies or other team members, rangefinder sensors are useful to detect long contiguous objects like walls.

B. Angle sensors and relative position sensors

Another kind of egocentric sensor is the angle sensor. This simply reports the angle to a particular object, or the nearest of some class of object. In the previously discussed car racing experiments, such sensors were used for sensing waypoints which were regularly spaced out on the track [123, 124, 126]. In their experiment with evolving combat bots for Quake III, Zanetti and Rhalibi [149] used angle sensors for positions of enemies and weapon pick-ups.

Related to this are relative position sensors, that report distances to some object along some pre-specified axes. For example, Yannakakis and Hallam [145] evolved ANNs to control the behaviour of cooperating ghosts for a clone of Pac-Man. In their setup each ANN receives the relative position to Pac-Man as input and the relative position of the closest ghost (specified by the distance along the x and y-axis).

C. Pathfinding Sensors

A type of sensor that is still considered egocentric but which does not take the orientation of the controlled agent into consideration, and instead takes the topology of the environment into consideration, is the pathfinding sensor. A pathfinding sensor reports the distance to the closest of some type of entity along the shortest path, as found with e.g. $A*$. (This distance is typically, but not always, longer than the distance along a straight line.) This is commonly used in 2D games. To take another Pac-Man example, Lucas evolved neural networks to play Ms. Pac-Man using distances along the shortest path to each ghost and to the nearest pill and power pill [71]. In his case, the controller was used as a state evaluator and the actual action selection was done using one-ply search.

Different input representations can also highly bias the evolutionary search and the type of strategies that can be discovered. For example, most approaches to learning Ms. Pac Man make a distinction between edible and threat ghosts. There is typically one sensor indicating the distance to the next edible ghost, being accompanied by a similar sensor for the closest threat ghost. The idea behind this separation is to make it easier for the ANN to evolve separate strategies to dealing with threatening and edible ghosts. While such a division makes it easier for traditional approaches to learn the game, they require domain specific knowledge that might not be available in all domains (e.g. edible ghosts are good, threat ghosts are bad) and might prevent certain strategies from emerging. For example, Schrum and Miikkulainen [106] who used the same sensors (i.e. unbiased sensors) for edible and threat ghosts, showed that a modular architecture (see section IV) allowed evolution to find unexpected behavioral divisions on its own. For example, a particular interesting behavior was that of luring ghosts near power pills, which would be harder to evolve with biased sensors. This example shows that there is a important relation between the evolutionary method and the type of sensors that it supports.

D. Third-person Input

In many games the evolved controller receives additional input beyond its first person sensors that is not tied to a specific frame of reference. For example, in games like Ms. Pac Man the controller typically receives the number of remaining normal and power pills or the number of edible/threat ghost [106]. Another example is including the current level of damage of the car in car racing game controllers [11, 67, 68].

In board games, in which the ANN does not directly control an agent (Section III-B), the neural network typically only receives third-person input. This could include such quantities as the piece difference and the type of piece occupying each square on the board in games like Chess [31], Checkers [18] or Go [69]. An important aspect in this context is the *geometry* of the particular domain. For example, by understanding the concept of adjacency in a game like Checkers (e.g. an opponent's piece can be captured by jumping over it into an unoccupied field), a controller can learn a general strategy rather than an action tied to a specific position on the board. While earlier attempts at evolving state evaluators did not directly take geometry into account, by representing each board position with two separate neurons in the game Go [69], Fogel et al. [31] showed that learning geometry is critical to generalization. This is true for Checkers as well. In Blondie24 each node in the first hidden layer of the ANN receives input from a different subsquare of the Checkers board (e.g. the first hidden node receives input from the top/right 3×3 subsquare of the board). The evolutionary algorithm evolves the connection weights from the board inputs to the actual subsquares and also between the inputs and the final output node. The idea behind representing the board state as a series

of overlapping subsquares is to make it easier for the ANN to discover independent local relationships within each subsquare that can then be combined in the higher levels of the network. The geometry can also be represented by a convolutional recurrent network that "scans" the board [103].

E. Learning from raw sensory data

An exciting promise for NE approaches is to learn directly from raw sensory data instead of low-dimensional and pre-processed information. This is interesting for several reasons. One is that it might help us understand what aspects of the game's visual space is actually important, and how it should be processed, through a form of ludic computational cognitive science. Another is that forcing games to only rely on the very same information the human player gets makes for a more fair comparison with the human player, and might lead to human-like agent behaviour. More speculatively, forcing controllers to use representations that are independent from the game itself might enable more general game playing skills to develop.

Early steps towards learning from less processed data were performed by Gallagher and Ledwich [35] in a simplified version of Pac-Man. In their approach the world was represented as a square centered around Pac-Man and the direct encoded weights of the network were optimized by evolutionary strategy. While their result demonstrated that it is possible to learn from raw data, their evolved controllers performed worse than in the experiment by Lucas [71] in which the shortest path distances from Pac-Man's current location to each ghost, the nearest maze junction and nearest power pill were given to the controller.

A similar setup in the game Super Mario was chosen by Togelius et al. [130]. In their setup the authors used two grid-like sensors to detect environmental obstacles and another one for enemies. If an obstacle or an enemy occupies one of the sensors, the corresponding input to the ANN would be set to 1.0. Togelius et al. [130] compared setups with 9 (3×3), 25 (5×5), 49 (7×7) sensors. The authors compared a HyperNEAT-like approach with a MLP-based controller and showed that the MLP-based controller performs best with the smallest sensory setup (21 inputs total). While larger setups should potentially provide more information about the environment, a direct encoding can apparently not deal with the increased dimensionality of the search space. The HyperNEAT-like approach, on the other hand, performs equally well regardless of the size of the input window and can scale to 101 inputs because it can take the regularities in the environment description into account. The results of Lucas [71] and Togelius et al. [130] suggest that there is an intricate relationship between the method and the number of sensors and the type of game they can be applied to.

HyperNEAT has also shown promise in learning from less processed data in a simplified version of the RoboCup soccer game called Keepaway [137]. Using a two-dimensional *bird's eye view* representation of the game, the authors showed that their approach was able to hold the ball longer than any previously reported results from TD-based methods like SARSA or NE methods like NEAT. Additionally the authors showed that the introduced bird's eye view representation allows changing the number of players without changing the underlying representation, enabling task transfer from 3 vs. 2 to 4 vs. 3 Keepaway without further training.

More recently Hausknecht et al. [48] compared how different NE methods (e.g. CMA-ES, NEAT, HyperNEAT) can deal with different input representation for general Atari 2600 game playing. The highest level and least general representation was called object representation, in which an algorithm would automatically identify game objects at runtime (based on object images manually identified a priori). The location of these entities was then directly provided to the evolving neural network. Similar to the work by Togelius et al. [130], the results by Hausknecht et al. [48] also indicate that while direct network encodings work best on compact and pre-processed object state representations, indirect-encodings like Hyper-NEAT are capable of learning directly from high-dimensional raw input data.

While learning from raw sensory data is challenging in two-dimensional games, it becomes even more challenging in three-dimensions. One of the reasons is the need for some kind of depth perception or other distance estimation, the other is the non-locality of perception: the whole changes when you look around. In one of the early experiments to learn from raw sensory data in a three-dimensional setting, Floreano et al. [29] used a direct encoding to evolve neural networks for a simulated car racing task. In their setup, the neural network receives first-person visual input from a driving simulator called Carworld. The network is able to perform active vision through two output neurons that allow it to determine its visual focus and resolution in the next time step. While the actual input to the ANN was limited to 5×5 pixels of the visual field, the evolved network was able to drive better or equal to well-trained human drivers. Active vision approaches have also been successfully applied to board games like Go, in which a "roving eye" can self-directedly traverse the board and stop where it thinks the next stone should be placed [118]. More recently, Koutník et al. [62] evolved an indirectly encoded and recurrent controller for car driving in TORCS, which learned to drive based on a raw 64×64 pixel image. That experiment featured an indirect encoding of weights analogous to the JPEG compression in images.

Parker and Bryant [85, 86] evolved an ANN to shoot a moving enemy in Quake II, by only using raw sensory information. In their setup the bot was restricted to a flat world in which only a band of 14×2 gray-scale pixels from the center of the screen was used as input to the network. However, although the network learned to attack the enemy, the evolved bots were shooting constantly and spinning around in circles, just slowing down minimally when an enemy appeared in their view field. While promising, the results are still far from the level of a human player or indeed one that has received more processed information such as angles and distances.

VIII. Open Challenges

This paper has presented and categorized a large body of work where NE has been applied, mostly but not exclusively in

the role of controlling a game agent or representing a strategy. There are many successes, and NE is already a technique that can be applied more or less out of the box for some problems. But there are also some domains and problems where we have not yet reached satisfactory performance, and other tasks that have not been attempted. There are also various NE approaches that have been only superficially explored. In the following, we list what we consider the currently most promising future research directions in NE. While there are plenty of basic research questions in evolutionary computation and NE, which are important for the application of these techniques in games, this section will mostly focus on applied research in the sense of research motivated by use in games.

A. Reaching Record-beating Performance

We have seen throughout the paper that NE performs very well in many domains, especially those involving some kind of continuous control. For some problems (see Section II-B1) NE is the currently best known method. Extending the range of problems and domains on which NE performs well is an important research direction in its own right. In recent years, Monte Carlo Tree Search (MCTS) has provided record-beating performance in many game domains [6]. It is likely that many clues can be taken from this new family of algorithms in order to improve NE, and there are probably also hybridisations possible. Of course, performance can be measured in many different ways; in game tasks, it is often (though not always) about playing well (for measure of good playing) within some given computation time limit.

B. Comparing and combining evolution with other learning methods

While NE is easily applicable and often high-performing, and sometimes the best approach available for some problem, it is almost never the only type of algorithm that can be applied to a problem. There are always other evolvable representations, such as expression trees used in genetic programming. For player modeling, supervised learning algorithms based on gradient descent can often be applied, and for reinforcement learning problems, one could choose to apply algorithms from the temporal difference learning family (e.g. TD(0) or Q-learning). The relative performance of alternative methods compared to Q-learning differs drastically; sometimes NE really is the best thing to use, sometimes not. The outstanding research question here is: when should one use NE?

From the few published comparative studies of NE with other kinds of reinforcement learning algorithms, we can learn that in many cases, TD-based algorithms learn faster but are more brittle and NE eventually reaches higher performance [74, 100, 142]. But sometimes, TD-learning performs very well when well tuned [73], and sometimes NE completely dominates all other algorithms [42]. What is needed is some sort of general theory of what problem characteristics advantage and disadvantage NE; for this, we need parameterizable benchmarks that will help us chart the problem space from the vantage point of algorithm performance [131]. There have been some attempts at constructing such benchmarks

previously [57], and the General Video Game Playing Competition characterizes its games according to game design characteristics (problem features), allowing another way of comparing performance on problem classes [88].

But mapping the relative strengths of these algorithms is really just the first step. Once we know when NE works better or worse than other algorithms, we can start inventing hybrid algorithms that combine the strengths of both neuroevolution and its alternatives, in particular TD-learning and GP. Previous research by Whiteson and Stone in combining NEAT and Q-learning has shown promising results [138, 139]. These methods have been applied with some success to shooters [89] and racing games [13].

C. Learning from high-dimensional/raw data

As discussed in Section VII-E, learning from raw images or similar high-dimensional unprocessed data is a hard challenge of considerable scientific interest and with several applications. The paucity of experiments in evolving neural networks that control game agents based on raw data is puzzling given the fertility of this topic. However, as we can see from the published results, it is also very hard to make this work. It stands to reason why the best results have been achieved using drastically scaled down first-person image feeds. Direct shallow approaches seem to be unable to deal with the dimensionality and signal transformation necessary for extracting high-level information from high-dimensional feeds. However, recent advances in convolutional networks and other deep learning architectures on one hand, and in indirect encodings like HyperNEAT on the other, promise significantly improved performance. It seems like very interesting work would result from the sustained application of these techniques to e.g. the visual inputs from Quake II. This kind of task might also be the catalyst for the development of new evolvable indirect neural network encodings.

D. General video game playing

One of the strengths of NE is how generic it is; the same algorithm can, with relatively few tweaks, be applied to a large number of different game-related tasks. Yet, almost all of the papers cited in this paper use only a single game each for a testbed. The problem of how to construct NE solutions that can learn to play any of a number of games is seriously understudied. One of the very few exceptions is the recent work on learning to play arbitrary Atari games [47, 48]. While NE performed admirably in that work, there is clearly scope for improvement. Apart from the Atari GGP benchmark, another relevant benchmark here is the General Video Game Playing Competition, which uses games encoded in the Video Game Description Language (VGDL) [27, 102]. This means that unlike the Atari GGP benchmark, this competition (and its associated benchmark software) can feature generated games, and thus a theoretically unbounded set of video games. A controller architecture that could be evolved to play any game from a large set would be a step towards more generic AI capabilities. The first edition of the competition was won by controllers based on variations of Monte Carlo Tree Search,

but future editions will feature a "learning track" which allows controllers considerable time to train on each game [88] – NE methods are likely to be competitive here.

E. Combining NE with life-long learning

An even larger step would to be evolve a single neural network that could *learn* and *adapt* during its lifetime (ontogenetically, i.e. without further evolution) to play one of a set of games. This has as far as we know never been attempted, but would be highly impressive. Evolution and learning are two forms of biological adaptation that operate on very different timescales. Learning can allow an organism to adapt much faster to environmental changes by modifying its behaviors during its lifetime. One way that NE can create such *adaptive ANNs* is to not only evolve the weights of an ANN but also local synaptic plasticity parameters that determine how the weights of the network change during the *lifetime* of the agent based on incoming activation [30, 113, 133, 134]. This resembles the way the brains of organisms in nature can cope with changing and unpredictable situations [49].

While there has been progress in this field, adaptive ANNs have so far mostly been applied to relatively simple toy problems. However, novel combinations of recent advances such as more advanced forms of local plasticity (e.g. neuromodulation [113]), hypothesis testing in distal reward learning [112], larger indirectly-encoded adaptive networks [91–93], methods that avoid deception inherent in evolving learning architectures [63, 94], and learning of large behavioral repertoires [23], could allow the creating of learning networks for more complex domains such as games.

Such adaptive networks could overcome many of the challenges in applying NE to games, such as adjusting on the fly to the difficulty of the opponent, incrementally learning new skills without forgetting current ones, and ultimately allow general video game playing ANNs. However, preventing these networks from potentially learning undesired behaviors in addition to being reliable and controllable, are important future research direction, especially in the context of commercial games (Section VIII-G).

F. Competitive and cooperative coevolution

In our discussion of fitness functions in Section VI, we discuss competitive and cooperative coevolution at some length. This is because these approaches bear exceptional promise. Competitive coevolution could in theory enable open-ended evolution through arms races; cooperative coevolution could help find solutions to complex problems through automatically decomposing the problems and evaluating partial solutions by how well they fit together. Unfortunately, various problems beset these approaches and prevent them from achieving their full potential. For cooperative coevolution there is the problem of how to select the appropriate level of modularisation, i.e. which are the units that are cooperatively coevolved. Competitive coevolution has several pathologies, such as cycling and loss of gradient. However, we suspect that these problems have as much to do with the benchmark as with the algorithm. For example, open-ended evolution might not be achievable in the

predator-prey scenarios that were used in previous research, as there is just no room for more sophisticated strategies. Modern games might provide the kind of environments that would allow more open-ended evolution to take place.

G. Fast and reliable methods for commercial games

This paper has been an overview of the academic literature on NE in games rather than of the uses of NE in the game industry, for the simple reason that we do not know of many examples of NE being part of published commercial games (with the exception of the commercial game *Creatures* [44] and indie titles such as GAR [46] and Petalz [95, 96]). Therefore, one key research problem is to identify which aspects of neural networks, evolutionary algorithms and their combination have hindered its uptake in commercial game AI and try to remedy this. For example, ANNs have mostly found their way into commercial games for data mining purposes, with a few exceptions of usage for NPC control in games such as *Black&White* or the car racing game *Colin McRae Rally 2*. Game developers often cite the lack of control and clarity as an issue when working with neural networks. Especially if the ANN can learn *on-line* while the game is being played (Section VIII-E), how can we make sure it does not suddenly kill an NPC character that is vital to the game's story? Additionally, if the NPCs can change their behavior, game balancing is more challenging and new types of debugging tools might become necessary. In the future, it will be important to address these challenges to encourage the wider adoption of promising NE techniques in the game industry.

REFERENCES

[1] H. A. Abbass. Pareto neuro-evolution: Constructing ensemble of neural networks using multi-objective optimization. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 3, pages 2074–2080. IEEE, 2003.

[2] A. Agapitos, J. Togelius, S. M. Lucas, J. Schmidhuber, and A. Konstantinidis. Generating diverse opponents with multiobjective evolution. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 135–142. IEEE, 2008.

[3] B. Al-Khateeb. *Investigating evolutionary checkers by incorporating individual and social learning, N-tuple systems and a round robin tournament*. PhD thesis, University of Nottingham, 2011.

[4] A. Botea, B. Bouzy, M. Buro, C. Bauckhage, and D. Nau. Pathfinding in games. *Dagstuhl Follow-Ups*, 6, 2013.

[5] S. Branavan, D. Silver, and R. Barzilay. Non-linear monte-carlo search in Civilization II. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, pages 2404–2410. AAAI Press, 2011.

[6] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[7] B. D. Bryant and R. Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the 2003 congress on evolutionary computation (CEC)*, volume 3, pages 2194–2201, 2003.

[8] P. Burrow and S. M. Lucas. Evolution versus temporal difference learning for learning to play Ms. Pac-Man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 53–60. IEEE, 2009.

[9] M. V. Butz and T. Lonneker. Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 317–324. IEEE, 2009.

[10] L. Cardamone, D. Loiacono, and P. L. Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186. ACM, 2009.

[11] L. Cardamone, D. Loiacono, and P. L. Lanzi. On-line neuroevolution applied to the open racing car simulator. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2622–2629. IEEE, 2009.

[12] L. Cardamone, D. Loiacono, and P. L. Lanzi. Applying cooperative coevolution to compete in the 2009 TORCS endurance world championship. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[13] L. Cardamone, D. Loiacono, and P. L. Lanzi. Learning to drive in the open racing car simulator using online neuroevolution. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(3):176–190, 2010.

[14] L. Cardamone, A. Caiazzo, D. Loiacono, and P. L. Lanzi. Transfer of driving behaviors across different racing games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 227–234. IEEE, 2011.

[15] A. B. Cardona, J. Togelius, and M. J. Nelson. Competitive coevolution in Ms. Pac-Man. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1403–1410. IEEE, 2013.

[16] K. Chellapilla and D. B. Fogel. Evolving neural networks to play checkers without relying on expert knowledge. *Neural Networks, IEEE Transactions on*, 10(6):1382–1391, 1999.

[17] K. Chellapilla and D. B. Fogel. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, 87(9):1471–1496, 1999.

[18] K. Chellapilla and D. B. Fogel. Evolving an expert checkers playing program without using human expertise. *Evolutionary Computation, IEEE Transactions on*, 5(4):422–428, 2001.

[19] S. Y. Chong, M. K. Tan, and J. D. White. Observing the evolution of neural networks learning to play the game of Othello. *Evolutionary Computation, IEEE Transactions on*, 9(3):240–251, 2005.

[20] J. Clune and H. Lipson. Evolving 3D objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life (Alife-2011)*, volume 5, pages 2–12, New York, NY, USA, Nov. 2011. ACM.

[21] J. Clune, J.-B. Mouret, and H. Lipson. The evolutionary origins of modularity. *Proceedings of the Royal Society of London B: Biological Sciences*, 280(1755):20122863, 2013.

[22] C. A. C. Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary algorithms for solving multi-objective problems*, volume 242. Springer, 2002.

[23] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553): 503–507, 2015.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6 (2):182–197, 2002.

[25] G. A. Dubbin and K. O. Stanley. Learning to dance through interactive evolution. In *Applications of Evolutionary Computation*, pages 331–340. Springer, 2010.

[26] R. C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 81–86. IEEE, 2001.

[27] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius. Towards a video game description language. *Dagstuhl Follow-Ups*, 6, 2013.

[28] D. Floreano and F. Mondada. Evolution of plastic neurocontrollers for situated agents. In *From Animals to Animats 4, Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB" 1996)*, pages 402–410. MA: MIT Press, 1996.

[29] D. Floreano, T. Kato, D. Marocco, and E. Sauser. Coevolution of active vision and feature selection. *Biological cybernetics*, 90(3):218–228, 2004.

[30] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[31] D. B. Fogel. *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann, 2001.

[32] D. B. Fogel, T. J. Hays, S. Hahn, and J. Quon. A self-learning evolutionary chess program. *Proceedings of the IEEE*, 92(12):1947–1954, 2004.

[33] C. M. Fonseca, P. J. Fleming, et al. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *ICGA*, volume 93, pages 416–423, 1993.

[34] B. Fullmer and R. Miikkulainen. Evolving finite state behavior using marker-based genetic encoding of neural

networks. In *Proceedings of the First European Conference on Artificial Life. Cambridge, MA: MIT Press*, 1992.

[35] M. Gallagher and M. Ledwich. Evolving Pac-Man players: Can we learn from raw input? In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 282–287. IEEE, 2007.

[36] L. Galway, D. Charles, and M. Black. Machine learning in digital games: a survey. *Artificial Intelligence Review*, 29(2):123–161, 2008.

[37] J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural computation*, 22(7):1860–1898, 2010.

[38] J. Gauci and K. O. Stanley. Indirect encoding of neural networks for scalable Go. In *Parallel Problem Solving from Nature, PPSN XI*, pages 354–363. Springer, 2010.

[39] J. Gemrot, R. Kadlec, M. Bída, O. Burkert, R. Píbil, J. Havlíček, L. Zemčák, J. Šimlovič, R. Vansa, M. Štolba, et al. Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In *Agents for Games and Simulations*, pages 1–15. Springer, 2009.

[40] F. A. Gers and J. Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *Neural Networks, IEEE Transactions on*, 12 (6):1333–1340, 2001.

[41] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[42] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *The Journal of Machine Learning Research*, 9:937–965, 2008.

[43] F. J. Gomez, D. Burger, and R. Miikkulainen. A neuro-evolution method for dynamic resource allocation on a chip multiprocessor. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 4, pages 2355–2360. IEEE, 2001.

[44] S. Grand, D. Cliff, and A. Malhotra. Creatures: Artificial Life Autonomous Software Agents for Home Entertainment. In *Proceedings of the 1st International Conference on Autonomous Agents*, AGENTS'97, pages 22–29, New York, NY, USA, 1997. ACM. ISBN 0-89791-877-0. doi: 10.1145/267658.267663.

[45] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[46] E. J. Hastings, R. K. Guha, and K. O. Stanley. Automatic content generation in the galactic arms race video game. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(4):245–263, 2009.

[47] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone. HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player. In *Genetic and Evolutionary Computation Conference (GECCO) 2012*, 2012.

[48] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. A neuroevolution approach to general Atari game playing. In *IEEE Transactions on Computational Intelligence and AI in Games*, 2013.

[49] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, 1949.

[50] P. Hingston. A new design for a Turing test for bots. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 345–350. IEEE, 2010.

[51] A. K. Hoover, P. A. Szerlip, M. E. Norton, T. A. Brindle, Z. Merritt, and K. O. Stanley. Generating a complete multipart musical composition from a single monophonic melody with functional scaffolding. In *International Conference on Computational Creativity*, page 111, 2012.

[52] A. K. Hoover, P. A. Szerlip, and K. O. Stanley. Generating a complete multipart musical composition from a single monophonic melody with functional scaffolding. In M. L. Maher, K. Hammond, A. Pease, R. P. Y. Perez, D. Ventura, and G. Wiggins, editors, *Proceedings of the 3rd International Conference on Computational Creativity (ICCC-2012)*, 2012.

[53] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[54] E. J. Hughes. Piece difference: Simple to evolve? In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2470–2473. IEEE, 2003.

[55] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2588–2595. IEEE, 2003.

[56] D. Jallov, S. Risi, and J. Togelius. EvoCommander, 2015. URL http://http://jallov.com/thesis/.

[57] S. Kalyanakrishnan and P. Stone. Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1-2):205–247, 2011.

[58] I. V. Karpov, V. K. Valsalam, and R. Miikkulainen. Human-assisted neuroevolution through shaping, advice and examples. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO 2011)*, Dublin, Ireland, July 2011.

[59] G. M. Khan and J. F. Miller. Evolution of cartesian genetic programs capable of learning. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 707–714. ACM, 2009.

[60] G. M. Khan and J. F. Miller. In search of intelligence: evolving a developmental neuron capable of learning. *Connection Science*, 26(4):1–37, 2014.

[61] N. Kohl and R. Miikkulainen. Evolving neural networks for strategic decision-making problems. *Neural Networks*, 22(3):326–337, 2009.

[62] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 1061–1068. ACM, 2013.

[63] J. Lehman and R. Miikkulainen. Overcoming deception in evolution of cognitive behaviors. In *Proceedings of the Genetic and Evolutionary Computation Conference*

(GECCO 2014), Vancouver, BC, Canada, July 2014.

[64] A. Liapis, G. N. Yannakakis, and J. Togelius. Adapting models of visual aesthetics for personalized content creation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(3):213–228, 2012.

[65] A. Liapis, H. P. Martínez, J. Togelius, and G. N. Yannakakis. Transforming exploratory creativity with DeLeNoX. In *Proceedings of the Fourth International Conference on Computational Creativity*, pages 56–63, 2013.

[66] A. Lockett and R. Miikkulainen. Evolving opponent models for Texas Hold 'Em. In *2008 IEEE Conference on Computational Intelligence in Games*, December 2008.

[67] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez. The WCCI 2008 simulated car racing competition. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 119–126. IEEE, 2008.

[68] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lonneker, L. Cardamone, D. Perez, Y. Sáez, et al. The 2009 simulated car racing championship. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):131–147, 2010.

[69] A. Lubberts and R. Miikkulainen. Co-evolving a Go-playing neural network. In *Coevolution: Turning Adaptive Algorithms Upon Themselves, Birds-of-a-Feather Workshop, Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 6, 2001.

[70] S. Lucas and T. Runarsson. Preference learning for move prediction and evaluation function approximation in Othello. *Transaction on Computational Intelligence and AI in Games*, 6, 2014.

[71] S. M. Lucas. Evolving a neural network location evaluator to play Ms. Pac-Man. In G. Kendall and S. Lucas, editors, *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG 2005)*, pages 203–210. IEEE, 2005.

[72] S. M. Lucas and G. Kendall. Evolutionary computation and games. *Computational Intelligence Magazine, IEEE*, 1(1):10–18, 2006.

[73] S. M. Lucas and T. P. Runarsson. Temporal difference learning versus co-evolution for acquiring othello position evaluation. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 52–59. IEEE, 2006.

[74] S. M. Lucas and J. Togelius. Point-to-point car : an initial study of evolution versus temporal difference learning. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

[75] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. *Advanced Robotics*, 22(13-14):1521–1537, 2008.

[76] R. Miikkulainen. Creating intelligent agents in games. *The Bridge*, pages 5–13, 2006.

[77] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong. Computational intelligence in games. *Computational Intelligence: Principles and Practice*, pages 155–191, 2006.

[78] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong. Computational intelligence in games. In G. Y. Yen and D. B. Fogel, editors, *Computational Intelligence: Principles and Practice*. IEEE Computational Intelligence Society, Piscataway, NJ, 2006.

[79] D. E. Moriarty and R. Miikkulainen. Evolving neural networks to focus minimax search. In *AAAI*, pages 1371–1377, 1994.

[80] D. E. Moriarty and R. Miikkulainen. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3-4):3–4, 1995.

[81] H. Muñoz-Avila, C. Bauckhage, M. Bida, C. B. Congdon, and G. Kendall. Learning and game AI. *Dagstuhl Follow-Ups*, 6, 2013.

[82] S. Nolfi and D. Floreano. Coevolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artificial life*, 4(4):311–335, 1998.

[83] S. Nolfi and D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[84] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis. Imitating human playing styles in Super Mario Bros. *Entertainment Computing*, 4(2):93–104, 2013.

[85] M. Parker and B. D. Bryant. Neuro-visual control in the Quake II game engine. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 3828–3833. IEEE, 2008.

[86] M. Parker and B. D. Bryant. Neurovisual control in the Quake II environment. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):44–54, 2012.

[87] C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling player experience for content creation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):54–67, 2010.

[88] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, 2015.

[89] J. Reeder, R. Miguez, J. Sparks, M. Georgiopoulos, and G. Anagnostopoulos. Interactively evolved modular neural networks for game agent control. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 167–174. IEEE, 2008.

[90] N. Richards, D. Moriarty, and R. Miikkulainen. Evolving neural networks to play Go. In T. B"ack, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-97, East Lansing, MI)*, pages 768–775. San Francisco, CA: Morgan Kaufmann, 1998.

[91] S. Risi and K. O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *From Animals to Animats 11*, pages 533–543. Springer, 2010.

[92] S. Risi and K. O. Stanley. A unified approach to evolv-

ing plasticity and neural geometry. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[93] S. Risi and K. O. Stanley. Guided self-organization in indirectly encoded and evolving topographic maps. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2014). New York, NY: ACM (8 pages)*, 2014.

[94] S. Risi, C. E. Hughes, and K. O. Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491, 2010.

[95] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley. Combining search-based procedural content generation and social gaming in the Petalz video game. In *Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2012.

[96] S. Risi, J. Lehman, D. B. D'Ambrosio, and K. O. Stanley. Automatically categorizing procedurally generated content for collecting games. In *Proceedings of the Workshop on Procedural Content Generation in Games (PCG) at the 9th International Conference on the Foundations of Digital Games (FDG-2014). ACM, New York, NY, USA*, 2014.

[97] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley. Petalz: Search-based procedural content generation for the casual gamer. *Computational Intelligence and AI in Games, IEEE Transactions on*, PP(99): 1–1, 2015. ISSN 1943-068X. doi: 10.1109/TCIAIG. 2015.2416206.

[98] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):19S8.

[99] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5 (1):1–29, 1997.

[100] T. P. Runarsson and S. M. Lucas. Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. *Evolutionary Computation, IEEE Transactions on*, 9(6):628–640, 2005.

[101] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.

[102] T. Schaul. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[103] T. Schaul and J. Schmidhuber. Scalable neural networks for board games. In *Artificial Neural Networks–ICANN 2009*, pages 1005–1014. Springer, 2009.

[104] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by Evolino. *Neural computation*, 19(3):757–779, 2007.

[105] J. Schrum and R. Miikkulainen. Constructing complex NPC behavior via multi-objective neuroevolution. *AI-IDE*, 8:108–113, 2008.

[106] J. Schrum and R. Miikkulainen. Evolving multimodal behavior with modular neural networks in ms. pac-man. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014)*, Vancouver, BC, Canada, July 2014.

[107] J. Schrum, I. V. Karpov, and R. Miikkulainen. UT2: Human-like Behavior via Neuroevolution of Combat Behavior and Replay of Human Traces. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2011)*, pages 329–336, Seoul, South Korea, September 2011. IEEE.

[108] J. Schrum, I. V. Karpov, and R. Miikkulainen. Human-like combat behaviour via multiobjective neuroevolution. In *Believable Bots*, pages 119–150. Springer, 2012.

[109] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 19(3):373–403, 2011.

[110] N. Shaker, G. N. Yannakakis, and J. Togelius. Towards automatic personalized content generation for platform games. In *Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2010.

[111] N. Shaker, J. Togelius, and M. Nelson. Procedural content generation in games: A textbook and an overview of current research, 2015. To appear.

[112] A. Soltoggio. Short-term plasticity as causeeffect hypothesis testing in distal reward learning. *Biological Cybernetics*, 109(1):75–94, 2015. ISSN 0340-1200.

[113] A. Soltoggio, P. Dürr, C. Mattiussi, and D. Floreano. Evolving Neuromodulatory Topologies for Reinforcement Learning-like Problems. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, 2007.

[114] E. J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

[115] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[116] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[117] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *J. Artif. Intell. Res.(JAIR)*, 21:63–100, 2004.

[118] K. O. Stanley and R. Miikkulainen. Evolving a roving eye for Go. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag.

[119] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *Evolutionary Computation, IEEE Transactions on*, 9(6): 653–668, 2005.

[120] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[121] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[122] M. E. Taylor, S. Whiteson, and P. Stone. Transfer

via inter-task mappings in policy search reinforcement learning. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 156–163, May 2007.

[123] J. Togelius and S. M. Lucas. Evolving controllers for simulated car racing. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1906–1913. IEEE, 2005.

[124] J. Togelius and S. M. Lucas. Evolving robust and specialized car racing skills. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1187–1194. IEEE, 2006.

[125] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 111–118. IEEE, 2008.

[126] J. Togelius, P. Burrow, S. M. Lucas, et al. Multi-population competitive co-evolution of car racing controllers. In *IEEE Congress on Evolutionary Computation*, pages 4043–4050, 2007.

[127] J. Togelius, R. De Nardi, and S. M. Lucas. Towards automatic personalised content creation for racing games. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 252–259. IEEE, 2007.

[128] J. Togelius, S. Lucas, H. D. Thang, J. M. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum, et al. The 2007 IEEE CEC simulated car racing competition. *Genetic Programming and Evolvable Machines*, 9(4):295–329, 2008.

[129] J. Togelius, T. Schaul, J. Schmidhuber, and F. Gomez. Countering poisonous inputs with memetic neuroevolution. In *Parallel Problem Solving from Nature–PPSN X*, pages 610–619. Springer, 2008.

[130] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber. Super Mario evolution. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 156–161. IEEE, 2009.

[131] J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber. Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, 23(3): 30–33, 2009.

[132] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: a taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:172–186, 2011.

[133] P. Tonelli and J.-B. Mouret. On the relationships between generative encodings, regularity, and learning abilities when evolving plastic artificial neural networks. *PloS one*, 8(11):e79138, 2013.

[134] J. Urzelai and D. Floreano. Evolution of Adaptive Synapses: Robots with Fast Adaptive Behavior in New Environments. *Evolutionary Computation*, 9(4):495–524, 2001.

[135] N. van Hoorn, J. Togelius, and J. Schmidhuber. Hierarchical controller learning in a first-person shooter. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 294–301. IEEE, 2009.

[136] N. van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber. Robust player imitation using multiobjective evolution. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 652–659. IEEE, 2009.

[137] P. Verbancsics and K. O. Stanley. Evolving static representations for task transfer. *The Journal of Machine Learning Research*, 11:1737–1769, 2010.

[138] S. Whiteson and P. Stone. On-line evolutionary computation for reinforcement learning in stochastic domains. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1577–1584. ACM, 2006.

[139] S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.

[140] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving Keepaway soccer players through task decomposition. *Machine Learning*, 59(1):5–30, May 2005.

[141] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl. Automatic feature selection in neuroevolution. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1225–1232. ACM, 2005.

[142] S. Whiteson, M. E. Taylor, and P. Stone. Empirical studies in action selection with reinforcement learning. *Adaptive Behavior*, 15(1):33–50, 2007.

[143] S. Whiteson, M. E. Taylor, and P. Stone. Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 21(1):1–35, 2010.

[144] G. Yannakakis and J. Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and Games*, 2014.

[145] G. N. Yannakakis and J. Hallam. Evolving opponents for interesting interactive computer games. *From animals to animats*, 8:499–508, 2004.

[146] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André. Player modeling. *Dagstuhl Follow-Ups*, 6, 2013.

[147] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[148] X. Yao and M. M. Islam. Evolving artificial neural network ensembles. *Computational Intelligence Magazine, IEEE*, 3(1):31–42, 2008.

[149] S. Zanetti and A. E. Rhalibi. Machine learning techniques for FPS in Q3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '04, pages 239–244, New York, NY, USA, 2004. ACM. ISBN 1-58113-882-2.

[150] T. Ziemke. Remembering how to behave: Recurrent neural networks for adaptive robot behavior, 1999.