POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master's Degree
in Cinema and Media Engineering

Master's Degree Thesis

A Systematic Literature Review of Neuroevolution in Games

Supervisor

Prof. Marco Mazzaglia

Candidate

Roberto Piccolo

Academic Year 2023/2024

Abstract

Games have always been a driver for innovation in Artificial Intelligence technology. Evolving neural networks through evolutionary algorithms, also known as neuroevolution, proved to be a powerful and versatile tool in the field of game development. This systematic literature review (SLR) aims to provide a comprehensive overview of recent developments in the implementation of neuroevolution in the context of games.

Existing research is categorized in several broad themes such as optimal game-playing, procedural content generation, automated playtesting, behavior imitation and dynamic difficulty adjustment. Implementation techniques including Artificial Neural Networks architecture, fitness function design and evolutionary algorithm choice are investigated with the goal of identifying themes and patterns.

Furthermore, this SLR discusses open challenges and potential future directions for research in the domain of neuroevolution in games.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
ALE	Arcade Learning Environment
ANN	Artificial Neural Network
CES	Canonical Evolutionary Strategy
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CNN	Convolutional Neural Network
CPPN	Compositional Pattern Producing Network
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
EA	Evolutionary algorithm
ES	Evolution Strategy
FNN	Feed-forward Neural Network
FPS	First Person Shooter
GA	Genetic Algorithm
GAN	Generative Adversarial Network
IDVQ	Increasing Dictionary Vector Quantization
IIG	Imperfect Information Game
LSTM	Long Short-Term Memory
MAP-Elites	Multi-dimensional Archive of Phenotypic Elites
MCTS	Monte Carlo Tree Search
MLP	Multi-Layer Perceptron
NCA	Neural Cellular Automata
NEAT	Neuroevolution of Augmenting Topologies

NES	Nintendo Entertainment System
NPC	Non-Player Character
NSGA-II	Non-dominated Sorting Algorithm II
PCG	Procedural Content Generation
PfE	Play for Experience
PGS	Policy Gradient Search
PM	Player Modeling
PPO	Proximal Policy Optimization
PtW	Play to Win
ReLU	Rectified Linear Unit
RHEA	Rolling Horizon Evolutionary Algorithm
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RQ	Research Question
RTS	Real Time Strategy
SGD	Stochastic Gradient Descent
SLR	Systematic Literature Review
SMB	Super Mario Bros.
SNES	Super Nintendo Entertainment System
TWEANN	Topology and Weight Evolving Artificial Neural Network

1. Introduction

This review focuses on understanding neuroevolution's relationship with games in current research. Neuroevolution is a machine learning paradigm that adopts Evolutionary Algorithms (EAs) to optimize an Artificial Neural Network's (ANNs) parameters.

Artificial Intelligence (AI) has had a role in digital games since their inception, from primitive rules-based scripts to the more sophisticated Goal Oriented Action Planning and Behavior Trees [1]. While ANNs also can take the role of decision makers, these methods are rarely used in the development of commercial games. A possible reason for this (outside of computational budget) could be associated with ANNs being often described as black boxes in terms of their interpretability, making it difficult for developers to control the learned behavior to fulfill a specific design.

In an attempt to investigate how neuroevolution can be leveraged in the context of developing both digital and non-digital games, this study performs a systematic literature review of academic research on the topic. Three main Research Questions (RQs) are formulated to address this problem, focusing on how neuroevolutionary solutions are structured within this domain, what potential use cases exist, and what impacts result from adopting this technology. This approach aims at defining what neuroevolution can currently do and find underdeveloped possibilities, informing on open challenges.

A survey on the topic of neuroevolution in games curated by Risi and Togelius [2] exists and was last updated in 2017. Therefore, this review will focus on recent developments spanning the years between 2017 and 2023. The search strategy involves query-based interrogation of multiple digital databases, eliminating duplicates, and applying inclusion and exclusion criteria. Data extraction from the resulting work pool of 118 studies will classify the solutions based on their implementation, analyzing the parameters of Artificial Neural Networks (ANNs) in terms of input and output representation, starting network topology, and choice of evolutionary algorithm, including fitness function design. The studies will then be classified through a taxonomy derived from the possible roles AI can cover in games, as described in [1]. This is intended to identify themes and patterns between high-level goals and technical implementations.

The resulting macro categories include game-playing agents aiming to achieve high performance in the game domain, game-playing agents aiming to provide players with specific experiences, procedural content generation of gameplay-related elements, and player modeling. Each category is further divided into subcategories, except for player modeling, which is not large enough to warrant further division. Play-to-win agents are clustered based on the game testbed, while play-for-experience agents are categorized based on the type of experience provided (such as bug-free for automated playtesting, dynamic difficulty adjustment, or specific Non-Player Character behavior). Procedural content generation is categorized based on the type of element being generated.

The document is structured as follows. Next chapter (Chapter II) provides background information on foundational concepts for the review concerning machine learning, ANNs and EAs. Chapter III details the review method, detailing the research questions, search strategy, inclusion and exclusion criteria, search outcomes and primary categorization. Chapter IV then focuses on the implementation data extraction, classifying the technical characteristics of the

various neuroevolutionary solutions. Chapters V to VIII discuss the previously mentioned macro categories, their subcategories and analyze their contributions to the RQs. Finally, Chapter IX will summarize the findings, concluding with a discussion over the answers for the RQs, evaluating open challenges and discussing limitations in this study's design. Furthermore, Appendix A contains the full table containing the complete information from which the various categorizations are derived.

2. Background

Yannakakis and Togelius [1] define the utility function as a measure of how good a specific representation is through approximation. It can assume different names in different contexts: fitness in evolutionary computation, reward in reinforcement learning, error in supervised learning.

Machine learning aims to find the maximum (or minimum) utility representation.

2.1 Artificial Neural Networks

ANNs are just one of many possible different representations of computational knowledge including grammars, graphs, trees, and others.

ANNs are inspired by the biological functioning of the human brain. The basic unit in the network is the artificial neuron. It has a varying number of inputs weighted by their synaptic strength and provides an output through an activation function. The activation function receives a weighted sum of the neuron's inputs with the addition of a bias. This defines ANNs as mathematical models which are widely used as function approximators in various fields. Common activation functions are the Gaussian activation function, the sigmoid-shaped activation function, and the rectified linear unit (ReLU) [1].

ANNs are made of interconnected neurons. How these connections are structured defines the ANN's topology. The most common approach is to divide neurons in layers [1].

There are two main categories of ANN topology: Feed-forward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs) [3].

In FNNs connections are all in the same direction from the input layer to the output layer. A simple FNN is the Multi-Layer Perceptron (MLP). MLPs are composed of an input layer, one or more hidden layers and an output layer. Furthermore, MLPs are also fully connected (each node or neuron connects to every node in the following layer) [1].

In RNNs connections can create loops and are usually involved in sequential or time-series data.

2.2 Evolutionary Algorithms

EAs are a class of optimization algorithms inspired by the principles of biological evolution. Optimization algorithms look for the maximum or minimum utility (fitness) value in the solution search space [1].

EAs optimize by keeping a population of solutions and evolving them through mutation and crossover, keeping the best (as in highest utility or fitness) individuals and discarding the worst, resembling Darwinian natural selection. Crossover produces a new solution (offspring) through the recombination of two parent individuals (solutions) and is potentially destructive (an offspring could possibly have lower fitness than its parents). How this is accomplished is strongly related to the representation. Neither crossover nor mutation are mandatory, but at least one of them or a combination of both is utilized in evolutionary algorithms.

There are many critical decisions that can affect algorithm performance, both from a computational and utility standpoint: choosing or designing a representation, a utility function, an evolutionary algorithm and tuning its parameters (population size, number of generations, mutation and crossover methods, fitness selection).

This process iterates over multiple generations until a desired amount of them is exhausted or other termination criteria are reached. Eiben and Smith [4] summarized it as follows:

- A population of individuals (candidate solutions) is initialized (often randomly).
- Each individual is evaluated by a utility (fitness) function.
- A selection based on fitness and other potential constraints will fuel the following generation.
- Two or more parents recombine to create a new individual and/or a single individual mutates.
- The new generation is evaluated as in step 2.

Novelty is provided by crossover and mutation operators, while selection increases the overall quality of the solutions.

The defining elements of an evolutionary algorithm listed in [4] are a representation of the individuals, a utility (fitness) function, a population size, a parent selection method, variation operator implementations for the selected encoding, a replacement method to create new generations, initialization procedure for the starting population and a termination condition for the evolutionary loop.

Representation in EAs is the combination of phenotypes and genotypes. Phenotypes are individual candidate solutions in the population, genotypes are their encodings. The search space in EAs is the genotype space, therefore the most fit solution is therefore the decoding of the best genotype.

Utility (or evaluation, or fitness) functions are the basis for selection and measure the quality of a candidate solution.

Populations are collections of genotypes. In most cases their size stays constant during the evolution process. A population's diversity is defined as the number of different solutions, which can be measured in a variety of ways. Often diversity is evaluated in fitness values, phenotypes, or genotypes.

Parent selection is in many cases probabilistic, allowing low fitness individuals a chance to be selected as parents for the next generation in order to reduce the risk of being stuck in local optimum.

Variation operators stochastically generate new candidate solutions from the current population. They differ on their arity: mutation is unary, and recombination is n-ary.

Replacement strategy (or survivor selection strategy, depending on the offspring/population size ratio) determines which individuals will make it to the next generation, since population size is in most cases constant. Unlike parent selection, in most applications this is a deterministic process.

Initialization typically consists of generating a randomized starting population.

Common termination conditions for the evolutionary loop can be bound to CPU time, a total number of fitness evaluations, small improvements (below a defined threshold) between generations for a specific number of iterations, and a diversity drop in the population.

The authors in [4] also provide a definition for the concepts of exploration and exploitation in evolutionary algorithms. These concepts relate to regions of the search space where new

individuals are generated: exploitation narrows the scope around already known high fitness individuals, while exploration broadens this space. A balance between these two is necessary to avoid inefficient search (excessive exploration) or premature convergence (excessive exploitation).

In [4] two population management models are described: generational and steady-state. In the generational model populations are replaced by their offspring, which can potentially be more than the population size and selected by their fitness. In the steady-state model the number of individuals replaced by the offspring is smaller than the population size and is proportional to it (generational gap).

Parent selection can be performed with different strategies such as fitness proportional selection (probability of mating based on fitness), ranking selection (probability of mating based on rank after fitness sorting), tournament selection. Tournament selection does not need awareness of the population's fitness globally and only need to rank subsets of individuals (usually two), which performs well on larger populations where evaluating or sorting by fitness could prove computationally expensive. It also does not need to quantify fitness, only to compare it between candidates making it a popular choice [4].

Likewise, multiple survivor selection methods have been developed, age-based, fitness-based or both. Fitness-based replacement strategies include GENITOR (worst individuals are replaced, prone to local optimum), elitism (prevents highest fitness individual from the current generation to be replaced by lower fitness offspring), (μ, λ) selection where more children than the population size are created every generation mixing age-based (every individual lives for one cycle) and fitness-based (the λ offspring are sorted by fitness and the first μ are selected) [4].

Two of the most prominent and widely used EAs are Genetic Algorithms (GA) and Evolution Strategies (ES). In its most basic implementation, a GA typically uses 1-bit string representation and employs both the crossover and mutation operators and parents are selected based on their fitness [4]. An ES on the other hand encodes real-valued vectors (although both GA and ES can adapt to multiple encodings), primarily uses mutation, selects parents stochastically and features a self-adaptation mechanism, meaning that the algorithm's parameters (the mutation step sizes) are coevolved and encoded in the solution, dynamically affecting the evolution process.

2.3 Neuroevolution

The evolution of artificial neural networks through the evolutionary class of algorithms is called neuroevolution. Traditional neuroevolution evolves a fixed network topology which is designed a priori. The evolution process then searches for the optimal connection weights. ANNs capable of evolving both their topology (by adding and removing nodes and connections) are named Topology and Weight Evolving Artificial Neural Networks (TWEANNs). Stanley and Mikkulainen introduced the Neuroevolution of Augmenting Topologies (NEAT) for TWEANNs in their 2002 work [5]. This method uses historical markings to track genes in order to address the competing conventions problem caused by having multiple possible permutations representing the same solution. Another important aspect of NEAT is the protection of innovation through speciation. Innovation (intended as the addition or removal of new nodes and connections) often leads to a decrease in fitness in the short-term. Without a protection mechanism, these individuals potentially can't survive for enough generations to develop into a high fitness solution.

2.4 Multi-Objective Optimization

Unlike single-objective optimization, where there is a well-defined search space, multiobjective optimization yields a set of solutions of equivalent quality due to conflicting objectives. These solutions are partially ordered, and the goal is to find optimal trade-offs between the conflicting objectives, resulting in a set known as the Pareto-optimal set. This set comprises solutions that are not dominated by any other solution and represents the best possible trade-offs. Multiobjective optimization problems typically involve minimizing or maximizing multiple objective functions subject to various constraints [6].

3. Method

A SLR examines scientific texts specific to a particular research area or theory, providing a comprehensive overview and a critical analysis of the topic. This allows to create relationships between different research works, identify patterns and themes, find gaps in the literature, and provide future directions as well as an objective summary.

To achieve so, an initial search will be conducted on different electronic databases, as well as the scholarly literature search engine Google Scholar. The resulting work pools will be merged and selected through the definition of inclusion and exclusion criteria, cleaning up potential duplicates.

3.1 Research Questions

This work aims to summarize and classify recent research concerning neuroevolution in games. To achieve this goal, the following RQs were formulated:

Table 3.1. Research Questions

No.	Research Question
RQ1	What are the most common neuroevolution techniques in the context of developing digital and non-digital games?
RQ2	What are the potential use cases for neuroevolution in the context of developing digital and non-digital games?
RQ3	What are the costs and benefits of utilizing neuroevolution in the context of developing digital and non-digital games?

3.2 Search Strategy

The electronic databases consulted for this review are:

- Web of Science
- Scopus
- IEEE Electronic Library
- arXiv
- ACM Digital Library
- Google Scholar

Risi and Togelius [2] provided a state of the art on the topic of neuroevolution in games published in 2015 and last updated in March 2017. Therefore, this work will focus on the development of research in this area of study following March 2017, up to the data collection date, which is October 2023.

The articles included must be written in the English language.

3.3 Search terms

Due to the broad nature of neuroevolution in a variety of fields and to obtain a manageable number of relevant results, the search query was refined to be ((neuroevolution) OR ("neuro-evolution")) AND (game). The query assessed titles, abstracts, and keywords to create the initial data pool on which screening for extraction based on inclusion and exclusion criteria will be performed.

The same query was also executed on Google Scholar, where results are obtained through a full-text search. The first 1000 out of 3010 total matches were evaluated.

To apply the defined inclusion criterion, it is necessary to define what games are for the purpose of this work. While game studies are a relatively young area of research, they produced various definitions of what a game is, making it difficult to be exhaustive. So, to make this SLR as comprehensive as possible, everything that uses existing board games or video games as well as custom environments stated to be designed as games by their authors has been included.

At the same time, studies presenting abstract game-theory problems were excluded. These criteria align with Risi and Togelius' [2] previous 2017 state of the art.

Furthermore, since video game production is a field encompassing different disciplines which may overlap with other industries (e.g. music, sound design, art, 3D modeling, animation etc.), only techniques that are unique to games were considered. For instance, procedural content generation will be considered, but it will be limited to game-related elements such as levels, maps, and characters.

Table 3.2. Inclusion/exclusion criteria

No.	Criterion
IC1	The study applies evolutionary algorithms to evolve artificial neural networks (neuroevolution) in the context of digital and non-digital games.
EC1	The study is an older or conference version of another paper.
EC2	The study presents traditional game-theory problems.

Table 3.3. Search results per database after IC1.

Database	Search results	Meeting IC1
Web of Science	42	33
Scopus	87	63
IEEE Electronic Library	33	27
arXiv	18	14
ACM Digital Library	14	12
Google Scholar	1000	133

After merging the search outcomes, eliminating duplicates, and filtering the remaining entries by the criteria in Table 3.2, a total of 118 items were analyzed conducting this review.

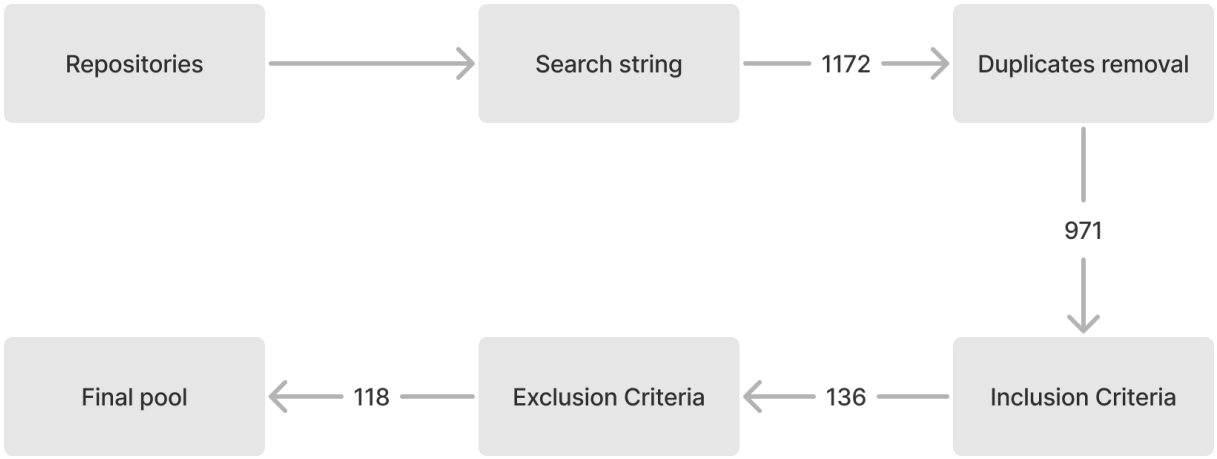

Figure 3.1: search process steps schematization.

3.4 Search Outcomes

Yannakakis and Togelius [1] identify 3 core game AI areas in their 2018 book "Artificial Intelligence and Games", based on the role played. The main goals AI can have in games are:

- Playing games, which can be further subdivided into playing games to win or achieve optimal performance and playing games to provide a desired user experience.
- Procedurally generate content.
- Model players through preference learning, behavior imitation or data mining.

A classification derived from this subdivision will be used to provide a first level categorization scheme, attempting to find answers for RQ2. The following percentages represent the distribution of studies in the proposed classification:

Table 3.4. First level categorization

N° of studies	%	Category
85	72.03	Play to win
14	11.86	Play for experience
14	11.86	Procedural content generation
5	4.24	Player modeling

Playing to win emerges as the most common goal for neuroevolution techniques among the studies under review. A variety of frameworks and environments use games as testbeds for AI algorithm research.

For each of these first level categories, a specific second level categorization has been designed.

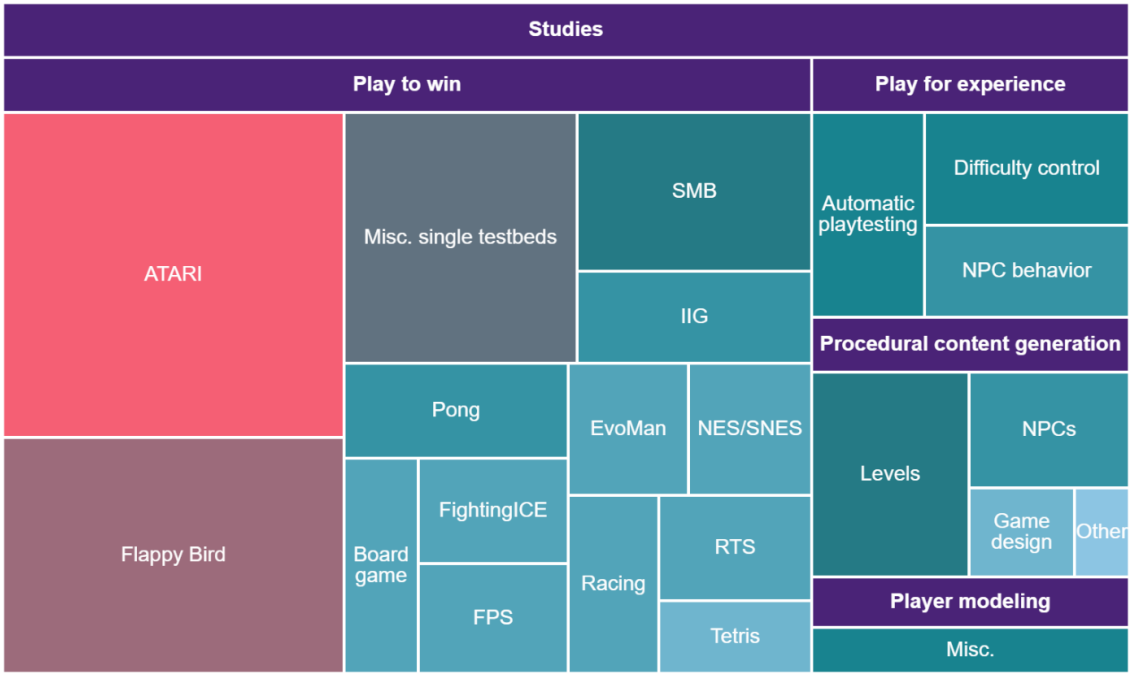

Figure 3.2: hierarchical view of first and second level categorization.

Play to win neuroevolutionary agent solutions are grouped by their testbeds. This allows for solutions to be evaluated within similar scope while also observing patterns in different types of games and game genres.

Studies in which the agent evolved through neuroevolution plays a game to help achieving a specific or improved experience are further grouped by the different desired experiences, such as a dynamic difficulty adjustment during gameplay (where a game's parameters influencing difficulty are tuned based on a player's measure of performance), automated playtesting (in order to provide players with a bug-free experience), and others.

Research work on procedural content generation can be subdivided into categories relating to the type of element being generated (e.g. levels, NPCs, game rules and designs).

There are not enough player modeling solutions to warrant further categorizations.

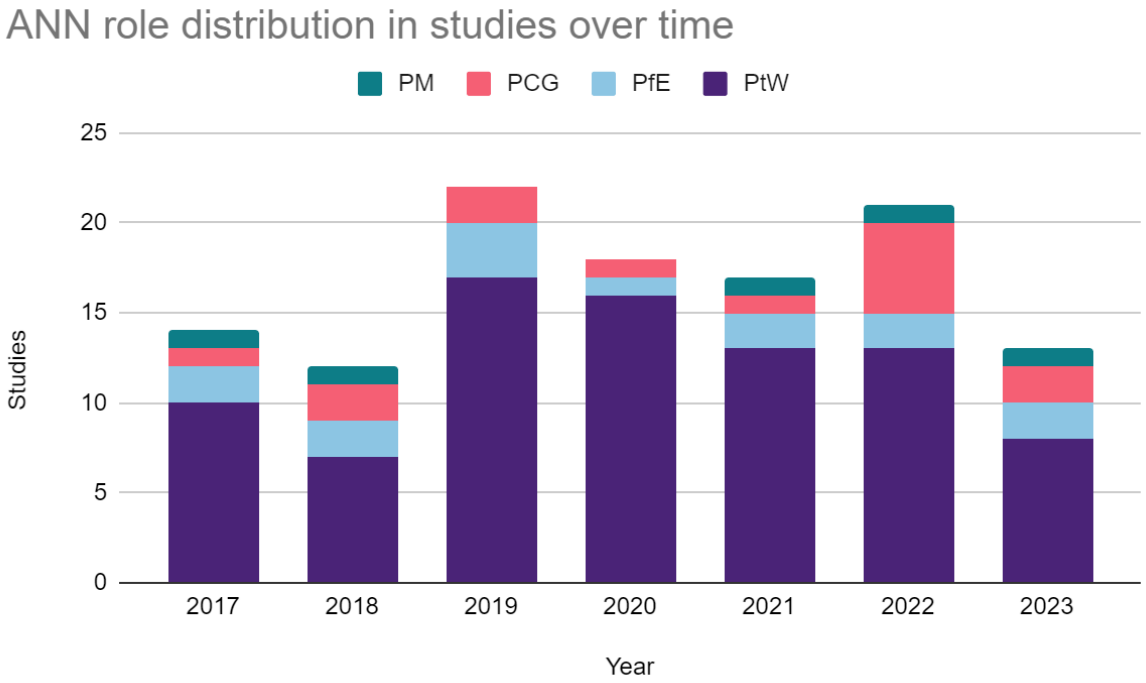

Figure 3.3: bar chart showing the distribution of studies over time based on first level categorization (PM = Player Modeling; PCG = Procedural Content Generation; PfE = Play for Experience; PtW = Play to Win).

The number of publications on the topic has remained relevant over the years, as depicted in Figure 3.1. There have been over 15 studies per year from 2019 onwards, except for 2023, which is limited to months ranging from January to October. The number of studies never drops below 12 (2018).

While these different goals can provide a good separation and aid in addressing RQ2, all these studies present solutions which share technical similarities in their implementation. At the same time, there are a lot of moving parts in neuroevolution between the ANN architecture of choice, the chosen evolutionary algorithm and its various parameters, the input/output mapping, the utility or fitness function design, as described in Chapter 2. In the following chapter, data extraction of technical implementation features will be shown and discussed, contributing to RQ1 and RQ3.

4. Implementations

This chapter analyzes the results of data extraction from the work pool in terms of the technical implementations of neuroevolutionary solutions. To do so, five main components are being examined: the ANNs' input and output representations, the starting network topologies, the fitness function design, and the chosen EA.

4.1 Input/output mapping

Observing the pool of studies examined in this review, it's possible to find a pattern in what functions as ANN input and output values and what these values mean in the context of function approximation.

4.1.1 Output

In the large majority of work, ANNs represent agents playing a game. The output of this kind of ANN is some kind of agent action representation. In environments where agents represent human players, this translates to mapping outputs to the possible actions the agents can take. In some custom engine-based environment cases, the possible actions are high level definitions of actions within the context of the game (e.g. "jump", "fire", etc.), while in all emulator-based frameworks such as the Arcade Learning Environment (ALE) it directly translates to controller inputs for that specific system.

Table 4.1. ANN output type distribution

Output type	N° of studies	%
Direct action selection	105	88.98
Content features	8	6.78
Indirect action selection	4	3.39
Function approximation	1	0.85

A few notable exceptions to this can be found in [7] and in [8]. The study in [7] study aims at leveraging neuroevolution with the goal of providing a good game balance in an asymmetrical Real Time Strategy (RTS) game specifically developed for this purpose. AI agents take the role of players, which in the context of this genre of games means that the decision-making impacts directing multiple units, each having its own diverse (if partially overlapping) set of possible actions. Instead of having AI agents output action selection directly for each unit, the output of ANN agents provides generic high level macro-actions. Based on the selected macro-action, the AI player will command traditional behavior-based AI agents to achieve the goals associated with it. Although the experiments run in this study show unsatisfying performance for this kind of solution, there's not enough detailed data to understand if the underlying cause is this high-level output design approach. It remains worth to consider further investigation on this topic, as there are multiple studies in this review that achieve positive benchmarks using a similarly modular high-level approach (combining neuroevolution and other techniques to target

different subtasks in the solution space) for compressing the input dimensionality (as opposed to output in [7]) which will be discussed further in this section.

[9] combines two ANNs: a self-supervised learning prediction ANN and a goal ANN for an agent playing the Doom First Person Shooter (FPS) videogame in the VizDoom environment. The prediction network learns to predict a small number of values representing the consequences of actions from a dataset. Key features like ammunition, health and enemy kills are the measurements and a goal vector weighs how desirable each of these goals is for the agent. A small evolving ANN then is fed measurements and provides indirect action selection through a goal vector output.

Comparing different emulator-based frameworks, like the already cited ALE, the Mario AI framework, BizHawk, and the EvoMan framework (while actually not being an emulator per se, its design and features mirror those of emulator-based framework, providing a testbed inspired to the NES action-platformer video game Mega Man II) it can be noticed that the dimensionality of output is low.

ATARI 2600 games feature 18 possible actions based on the controller. One action is the absence of input, another is a single button press. Eight actions correspond to the eight possible directions registered by the joystick, and the remaining eight actions result from the combination of each of the eight possible joystick directions with a button press [10].

The Mario AI framework encodes input (as in emulator input, not ANN inputs) as a 5-bit array. Each bit represents whether a button on the Nintendo Entertainment System (NES) controller has been pressed or not. The NES controller features 8 total buttons: a directional pad with 4 orthogonal directions and the A, B, Start and Select buttons. Since the videogame Super Mario Brothers does not use the up direction on the directional pad and since the start and select buttons do not provide functionalities that are useful towards reaching the game's goal of reaching the end of each level, only the left, right, down, A and B inputs are encoded in the array [11].

The Super Nintendo Entertainment System (SNES) controller features the same buttons as the NES with four additional available buttons.

The EvoMan framework lists five possible non-exclusive actions for the player actor: the left and right directional movement, shooting, jumping and releasing the jump button (this action is related to a common game mechanic in platform games, where characters can jump up to their full jump height by holding the jump button, and prematurely releasing the button results in a shorter jump) [12].

ANN output actions for agents playing the game Pong represent movement along an axis, which can range from one to four outputs depending on the representation. In [13] there are three actions for controlling the right paddle: up, down, no action. In [14] movement is reduced to a single value representing a shift over an axis (opposite signs for the value map to opposite directions, with 0 meaning no action) and there are multiple horizontal and vertical paddles during the evolutionary process.

Flappy Bird is also a very simple game, featuring a single possible jump action. Some clones introduce variations like [15], which introduces an additional dive action.

The VizDoom API supports all the original Doom videogame's keyboard and mouse actions for moving, shooting, and interacting with the environment [16]. Studies like [17] only use a subset of these actions in their experiments.

The StarCraft II API can be used to control units in Blizzard's RTS game StarCraft II. A unit's possible actions describe movement towards specific coordinates in the 2D game space and whether to attack a target.

Similarly, through the Brood War API (based on the RTS StarCraft: Broodwar) used in [18] ANN agents piloting units can select one of two options: action (which might translate to attack or heal depending on the unit's type, medic or marine) or evasion.

The most complex set of actions can be found in FightingICE agents. FightingICE is a 2D fighting game used as a testbed for the Fighting Game AI Competition [19]. Actors in this environment can perform 56 different actions, with only 40 of them being directly activatable by players, the others being consequences of other actions. The 40 actions can be triggered by different combinations of 7 keys: the A, B, C buttons, and the four directions up, down, left, right [20]. The study in [21] features a Deep Neural Network (DNN) solution using the full 56 actions set output. In [20] these actions get compressed into three values: two for the x and y coordinates where the action is directed and a third one to determine whether to move or perform an attack. This is further refined in [22] by the same authors, increasing the output number to 9 to add finer information on the type of offensive, defensive or movement action selected.

The environments discussed so far share a few similarities: they map ANN outputs to actions in game, often translating directly to controller inputs the same as a human player would input its own actions in a video game. In a few cases these actions are compressed in their number through different representations [14][20][22]. Outputs range from 1 to 18, with the notable exception of 56 in [21]. Even RTS games, which traditionally feature very complex states and interfaces, limit the application of neuroevolution to single units, achieving low dimensionality output.

Studies in the imperfect information game class of games (Blackjack and variations of Poker, more specifically) still map outputs to actions. The difference compared to the previous games is that actions do not represent a controller's input frame by frame in this case, but discrete actions within turn based games. The dimensionality remains contained, 3 actions for Poker (fold, call, raise) and 5 for Blackjack (surrender, stand, hit, double, split), although the action space for No Limit Texas Holdem is infinite and continuous as stated in [23].

In studies regarding board games like chess, dama, and Hnefatafl a few differences can be observed. All these games involve multiple pieces belonging to opposing players and positioned on a two-dimensional board. An action in the context of this type of game involves moving a piece to a new position and potentially capturing an opposing piece. This type of action therefore alters what is commonly called the board position. [24] (Hnefatafl, an ancient board game that originated in Northern Europe) and [25] (dama, or Turkish draughts, a checkers variant) ANN agents provide high dimensionality outputs representing probabilities for every possible move in the game space, with 588 output nodes for [24] and 256 in [25]. [26] takes a different approach, using the ANN as an approximator of the value of a given board position in chess, although experiments show worse performance (in win ratio) against GNU Chess, one of the first chess engines for Unix-based systems.

Moving forward to studies in the play for experience category, most solutions are still comparably low-dimensional. [27] takes an approach similar to [26], where the ANN's role is to serve as a function approximator for determining the value of moving towards a specific node in the context of the video game Ms PacMan, which features the player character navigating a maze (similarly to the original PacMan). This kind of approach moves the dimensionality from the output of the ANN to the iterator evaluating all the possible choices through the ANN approximator.

Work concerning automated playtesting focuses on NEATEST, an extension of the Whisker tool for MIT's visual coding environment targeted at students, Scratch [9][28][29]. Output of the ANN in [29] are generated from source code analysis as instances of a minimized set of actions (related to Scratch events such as key presses, mouse movement and click events, text input, sound input, and inaction), so their number may vary depending on the program being tested.

In procedural content generation we partially move away from ANN agents outputting game inputs. ANN-based generators map their outputs to representations of the type of element being generated. In most cases, a numerical encoding of elements within discrete structures such as tiles in 2D grids, voxels in 3D grids, type of enemies out of an existing set.

In [30] neuroevolution is used in procedural art generation in the form of assets within a virtual art gallery setting. Two different architectures of ANN are used for two-dimensional and three-dimensional assets. 2D asset-generating networks (for pictures) output hue, saturation, and value for a given pixel. 3D asset-generating networks (for sculptures) provide an additional output for the alpha channel of a voxel.

In [31] neuroevolution is combined with previously trained Generative Adversarial Networks (GANs) to generate large-scale levels. The role of evolved ANNs in this study is to define global patterns for rooms in a Zelda-style dungeon level (2D grid of interconnected rooms) and Super Mario Bros. (SMB) levels (sequences of tile-based adjacent level sections). The output provides information about the rooms/sections and depends on the game, 30 for SMB, 16 for Zelda: 10 for room-related information and 6 for information about connections with other rooms.

The study in [32] proposes an ANN acting as an enemy wave generator in a custom-built tower defense game (this type of strategy game requires the player to build and configure defenses against groups of enemies called "waves"). The output represents the priority for each kind of enemy in the game to be spawned, adding up to 6 different enemies.

There are a few cases in which neuroevolved agents playing a game are used for procedural content generation. One example is [33], which introduces Voxelbuild, a sandbox environment inspired by Minecraft where AI agents can build structures in a discrete 3D grid. Producing content for this kind of game is part of the goal of the game itself, hence why it is possible to achieve this goal by a game-playing agent (with a 6-dimensional output, aligned with other action space-based agents examined so far).

Another one is coevolution between agents and environment generators, as found in [34]. The agent ANN outputs actions, the level generator network (a neural cellular automaton) outputs values for a given tile in a 2D-grid environment (with values being mapped to different types of tiles such as walls, floor, doors, keys etc.). A similar coevolutionary approach is used to procedurally generate game rules and levels for endless runner games in [32], where level generators output tiles (6 possible values corresponding to different tile types), and game generators output an encoding of game rules designed as combinations of predefined flags and parameters (under 20 in size).

In procedural content generation of Non-Player Characters (NPCs) [35][36][37] the ANNs' role is the element being procedurally generated through evolution. Therefore, the output of such ANNs once again corresponds to their action space, which depends on the specific game for its size.

Research on player modeling through neuroevolution also features mostly game-playing agent ANNs, with the notable exception of [38]. In this study, a neuroevolved ANN takes the role of a preference learner designed to predict player arousal from annotated game footage and the output is a single scalar value.

To sum it up, the output for ANN evolved with neuroevolution in the examined studies provides direct action selection in the large majority of cases (often in the form of controller input). In procedural content generation where ANNs cover the role of generators directly, the output describes generated content features. A few approaches in game-playing agents provide indirect action selection, and the aforementioned study [38] is the only one featuring a pure function approximator output.

Output dimensionality is under 20 nodes for most studies in the research pool, with few exceptions: board game positions and FightingICE full action sets. Procedural content generation of maps and similar game elements offer scalable outputs. In some cases, the output is fixed, but the dimensionality of the solution involving the use of neuroevolution scales as a whole.

4.1.2 Input

Input representation is also dependent on the ANN role in the neuroevolution solution. As mentioned in the previous section, most networks function as game-playing agents. In order to make meaningful decisions in the form of action selection outputs the input must inform the ANN on the current state of the game. In literature, this is often referred to as observation [2]. From the examined work emerges a main distinction between two main different approaches in providing such observations of the game's current state to game-playing agents: raw visual input or game state extracted features.

Table 4.2. ANN input type distribution.

Input type	N° of studies	%
Game state	80	67.80
Visual frame	28	23.73
RAM	6	5.08
Coordinates	7	5.93

Each study can be categorized into multiple categories based on its content. As a result, the total percentage across all categories may exceed 100% because some studies contribute to multiple categories.

Raw visual input means that the agent is provided with screen image information in the form of a 2D grid of pixels with coordinates and color information. As mentioned in [2], this approach is independent of the game from an input perspective and is closer to general game playing. It is overall larger dimensionally than its game state features counterpart and comes with different challenges because of this, especially those featuring NEAT-based algorithms, as higher input dimensionality in growing ANN architectures affects computational resources, sometimes significantly, which will be addressed further down in this section.

Game state extracted features are information based on knowledge of the entities inside the game as perceived by the ANN agent or made available by the game itself. This type of input is smaller in dimensionality, requires access to a game's source code or relies on APIs for the game state data extraction. This also typically requires human curation in selecting meaningful features and designing appropriate sensors for agents. The authors in [2] provide a useful detailed taxonomy of different sensors an ANN agent can employ that is still accurate, such as

straight line and pie slice sensors (for the many games featuring entities in a two or three-dimensional space), or relative positions.

An interesting variation to game state features extracted during the simulation in real-time is found in the studies [29] and [39], where relevant gameplay logs are used to provide input information to ANN agents learning to play the game.

Other than these two main input representations, some emulator-powered solutions take advantage of having access to RAM states of running games. This makes it possible to have game state features extracted from memory positions rather than manually curated and exposed, either through source code manipulation or API leverage. RAM states provide generality in a similar way to raw screen inputs, but their dimensionality is also tied to the specific framework (more precisely, to the machine being emulated).

The Atari 2600 emulator provides visual frames in the form of a 160x210 7-bit pixels screen and accepts a discrete action space of 18 controller inputs. Furthermore, ALE allows access to the Atari 2600's 1024 bits of RAM, from which it is possible to extract game state features [10]. Mnih et al. [40] popularized a preprocessing method for Atari frames downscaling them to an 84x84 4-bit grayscale representation.

The authors in [41] annotated state variables from comments on disassembled code and source code for 22 Atari games. The study in [42] further categorizes games available in the framework by the quality of these annotations (based on the completeness of the information).

Most of the work in the ALE environment relies on visual frame input, with a few exceptions relying on RAM states. The other framework in which RAM states are used to extract input features is the Bizhawk multi-platform emulator, which studies [43] and [44] used to play NES and SNES games respectively.

[45] is seminal work which is often cited in other studies in the same cluster, where an ES approach is applied to Reinforcement Learning (RL) demonstrating the scalability of this kind of solution thanks to parallelization and its invariance to frame-skipping in the Atari testbed. Frame-skipping is common practice in visual input-based reinforcement learning tasks: to optimize the performance agents evaluate observations at a lower frequency compared to the simulation environment they run in.

Studies developed on custom engines or frameworks providing game state features tend to leverage those, except for those focusing on learning from visual input specifically. In [17], an autoencoder is used to compress the visual input from VizDoom [16]. The autoencoder trained with backpropagation receives 120x160x3 RGB pixel representation of the game and outputs a compressed version of it. [46] features another autoencoder solution in the ALE, trained alongside the ANN agent, to leverage direct encoding neuroevolution in high-dimensional domains.

In procedural content generation related to game levels, maps and such, studies often deal with rigid structures as described in the previous section. In these cases, the input provided carries the coordinates of a given unit in the examined rigid structure (2D or 3D grids, linear sequences etc.).

The authors in [47] (level generation for SMB) provide additional information about the tiles surrounding the one to be generated, with additional noise perturbation to make the process non-deterministic. A similar approach is shared by [48], which attempts to generate complex levels hierarchically through recursion in a Minecraft-like voxel environment, and [34], in which agents and generators are coevolved in order to generate levels for a 2D game inspired by The Legend of Zelda.

As mentioned in the previous section, in [32] procedural content generation for Voxelbuild is delegated to a game-playing agent's decision making within the sandbox. Therefore, the input representation is game state sensory data extracted by the agent. Conceptually, it compares to other level or structure generators, as the sensory data is represented by an 11x11x11 cube of blocks around the agent within its block placement range of 5.

In [32], where procedural content generation happens at runtime during the game, game state information is used as input for generating enemy waves in a tower defense game. This is interesting because it shares input information with game-playing agents. From this perspective, ANNs generating content with characteristics tied to game mechanics and rules are structurally similar to game-playing agents input wise.

It is interesting to find some kind of comparison between game state and visual frame-based input representations for neuroevolution in games. The study in [49] compares different evolutionary algorithms and different input dimensionalities, including different visual frame representations and low-dimensional game state inputs. Experiments show that the algorithms performed best (score wise) on game state low dimensionality input. Another comparison can be found in [50], where different encodings and different inputs are measured in the Tetris testbed. Experiments show direct encoding on game state features achieving the highest score compared to its indirect encoding and raw visual frame input data.

In summary, referencing Table 4.2, the most common input representation for game-playing agents is game state features, followed by visual frames. A few studies take advantage of RAM states exposed by emulators to extract game state features. Visual frames are mostly used in the ALE and provide better generality in game playing agents, as work in this framework often produces agents able to play a variety of games. They also come with higher dimensionality in input, with some solutions leveraging autoencoders or other means of preprocessing for dimensionality reduction, especially relevant in NEAT-based solutions, where the size of the network impacts the performance of the evolutionary process. Game state features are extracted during the simulation through a variety of sensors, with a couple exceptions using gameplay logs. Although not very numerous, there are experiments showing better performance (in score and computational costs) for game state inputs compared to visual frames.

4.2 Artificial Neural Network Architectures

After discussing input and output nodes, it might be useful to take a look at what is between them. As stated in Chapter II, there are two main branches of neural network topologies: feedforward and recurrent. These types of networks differ in how neurons are connected between layers. Feedforward networks have neurons connecting forwards from input to output, while recurrent networks can connect backwards and form loops, making them useful for sequences and retaining information from previous time steps.

The majority of work under review employs NEAT or NEAT-inspired TWEANN solutions, an evolutionary algorithm also discussed in Chapter II which is capable of altering a network's topology during the evolutionary process, therefore it can evolve both feedforward and recurrent neural networks, making it difficult to extract valuable information on this matter. It is still of interest to discuss starting networks (when described, as they influence the evolutionary process), nested architectures and patterns in how they are employed in the studies under review to help answer research questions. The most specific definition of architecture is used when available (e.g. LSTM cells are a subtype of RNNs, but are counted separately from generic RNNs; similarly DNNs may contain multiple layers).

Table 4.3. Starting ANN architectures distribution.

Starting Network	N° of studies	%
RNN	10	8.47
LSTM	7	5.93
CNN	3	2.54
CPPN	8	6.78
DNN	20	16.95
FNN	75	63.56

Each study can be categorized into multiple categories based on its content. As a result, the total percentage across all categories may exceed 100% because some studies contribute to multiple categories.

The following sections will analyze the studies grouped by their starting ANN of choice.

4.2.1 RNNs and LSTMs

In [51] a modular network (Context+Skill Model) is introduced. It consists of a skill module and a context module both receiving sensory input, and a controller model receiving the other two modules' output. The controller's output is the agent's action. Skill and controller module are FNNs, while the context module is a Long Short-Term Memory (LSTM). This memory cell allows the agent to learn sequences of actions. Experiments in multiple testbeds (modified versions of Flappy Bird, Lunar Lander and an autonomous driving simulation environment) show better generalization capabilities than context only or skill only networks, although it would be interesting to analyze this solution's behavior on different testbeds.

LSTMs are extensively used in imperfect information games to maintain historical information on opponents, which helps agents making decisions. [52] features 15 LSTM cells in a nested DNN setup combining neuroevolution and curriculum learning (learning through increasingly complex examples) for six-player poker. 10 LSTMs with hand information that resets every round are part of a game module, 5 LSTM networks for modeling opponent hands and history are part of the opponent module. Both modules feed their outputs to a FNN decision module. The agent is trained with a GA against a random subset of a set of bots with diverse playstyles, with no subset made of a single type of bot. Results from experiments show a higher fitness compared to baseline passive curricula.

Another poker opponent exploitation hybrid strategy using neuroevolution and RL can be found in [23]. The authors in this study measure critically improved rewards after introducing a LSTM layer between the input layer and the hidden layer to extract temporal features.

In [53], ensemble learners are used in Leduc poker playing (a simpler poker variant used in research). LSTM cells are also used for historical data representation and a MLP outputs actions.

The study in [54] uses neuroevolution in a similar context. A pattern recognition tree is added to two LSTM estimators to model Poker opponents.

[55] proposes a different context for LSTM usage. In this study, the temporal information is extracted from image frames instead of game state features unlike the aforementioned studies.

Again, this is part of a modular setup made of a variational autoencoder, a LSTM module for temporal features and a controller network for decision-making. Results from experiments run in a top-down racing game testbed showed that the predictive memory component had higher activation levels when the agent was near to a corner and about to turn.

[56] further explores the topic, featuring [55] as a baseline along other methods. In this study the goal is to create an agent able to leverage self-attention to determine important features for its task through visual hints as a form of dimensionality reduction supporting generalized game-playing over visual input.

A comparison within different RNN architectures can be found in [37], a study that attempts to procedurally generate a communication language for AI agents. Speakers and listeners are both represented by RNNs, with two main architectures being tested: Gated Recurrent Units (GRUs) and partially randomized RNNs. Both types of agents use a FNN to preprocess inputs for the RNN. Experiments run on a dataset on evolved and unevolved versions of both the ANN architectures showed better performance for their evolved counterparts, with GRUs performing better than partially randomized RNNs in the repeated reference game testbed. Still, neuroevolution performed worse than the deep reinforcement learning baseline in this benchmark.

4.2.2 CNNs

In [57] the author reports switching from a CNN architecture to a FNN one while developing an agent playing the Asteroids game in the Atari testbed. This went together with switching from visual input to game state features, showing largely improved performance in computational power usage.

The premise of the work in [58] is the research result of CNNs being proved superior to fully connected networks in deep learning research applied to visual inputs. The author proceeds to evolve CNNs with the indirect encoding algorithm HyperNEAT to develop game-playing agents for Tetris over visual frames, comparing them with fully connected versions. HyperNEAT evolves CPPNs to leverage symmetries and pattern repetitions in ANN encoding, basically compressing the ANN representation to overcome base NEAT's limitations with increasingly large ANN sizes (evolution process taking longer, numerous nodes making mutations overall less impactful). The study proposes a HyperNEAT variation for CNN encoding. CNNs feature fewer connections compared to fully connected networks. Experiments compare two different link encodings and coordinate geometries (multi-spatial substrates and global coordinates). Results show that the CNN-HyperNEAT variation is superior to its fully connected counterpart in the Tetris testbed, multi-spatial substrates outperform global coordinates, no clear differences are found between different link encodings. Deep networks perform poorly compared to shallow networks. Results from shallow networks are not record-breaking although, as higher scores have been achieved with high-level game state features.

4.2.3 CPPNs

Other instances of CPPNs associated with HyperNEAT usage can be found in [50], to which [58] is a follow-up, with experiments on visual input versus game state feature in Tetris. [49] features HyperNEAT (among other methods for comparison) and therefore CPPNs in the Atari testbed, also using its indirect encoding capabilities to address the high dimensionality of visual input.

A different usage, perhaps more "traditional", for CPPNs is common among procedural content generation studies. The Infinite Art Gallery in [30] evolves CPPNs to generate 2D and 3D artwork, leveraging the symmetries and patterns in their activation functions, which can be different for each neuron. In [47] CPPNs are evolved to encode Boolean lattices for Minecraft structures (where the truth value translates to empty or non-empty voxels) which become part of a larger solution where these structures are cleaned up and materials are assigned to voxels. [32] uses HyperNEAT and CPPNs to evolve agents building structures in the Voxelbuild environment. As stated in the previous Input section, these agents' observations consist of a 3D grid of blocks within their building range and are of high dimensionality, comparable to visual inputs (where it is more common to find indirect encoding solutions, as discussed in the further section concerning evolutionary algorithms).

The study in [31] and its follow-up [59] use CPPNs as generators for level layouts which are then fed to a generative adversarial network whose role is to generate specific level segments in a modular setup with the goal of creating large-scale levels for the 2D games Zelda and SMB.

4.2.4 DNNs

Typically, starting NEAT evolved networks feature no hidden layers, in order to evolve minimal size ANNs. This is common for the many listed game state input and action output network setups. Studies combining deep reinforcement learning and neuroevolution often feature deep starting neural networks. One such example is [60], where the initial population is made of DNNs initialized with random weights that are then trained with Deep Reinforcement Learning (DRL) and evolved with multi-objective optimization to promote diversity in a fighting game automatic playtesting scenario.

DNNs are also popular in visual input, especially in the Atari testbed. As stated in [61], it is a very popular architecture in the Atari 2600 testbed, which Chapter V will prove as a very prominent testbed for algorithmic research. One of the reasons can be found in the seminal work by Mnih et al. [41] in deep reinforcement learning, inspiring many other studies, also among those under review.

[62] takes a different approach on indirect encoding, evolving hypernetworks. Compared to HyperNEAT, hypernetworks do not require spatial features nodes and are trained with backpropagation. Compared to direct encoding in experiments over three different games there is no clear winner between direct and indirect encodings, although greater structure regularity can be observed in indirectly encoded networks.

To wrap up, NEAT-based by nature solutions cannot easily be categorized as purely recurrent or feedforward. It is still possible to discuss starting networks and fixed topology networks. LSTMs are useful in modular networks for imperfect information games, especially in poker opponent modeling and exploitation, but also find use in other game-playing agents, improving decision-making with temporal features. CPPNs are employed in procedural content generation for their geometric features and are also part of HyperNEAT-based indirect encoding solutions. Deep and convolutional networks are often used with visual input and when combining DRL with neuroevolution. Low dimensionality NEAT-evolved ANNs typically start from feedforward networks, often without hidden layers, and in many cases represent simple game state features to agent action mappings in game-playing agents.

4.3 Fitness functions

Another critical component in evolutionary algorithms is the fitness function, as described in Chapter II. Similarly to output representations, there's a strong connection between an ANN's role in the solution and how fitness is evaluated. The goal in this section is to understand what and how influences fitness function design and its relation to the exploration versus exploitation trade-off problem in evolutionary algorithms.

The main contributions to fitness functions can be classified as follows:

- In-game performance (measure of a game-playing agent's competency in reaching a game's main goal or goals)
- Off-game performance (measure of a game-playing agent's performance related to elements that do not belong to the game e.g. how closely it imitates a given behavior)
- Content features (characteristics of procedurally generated content e.g. solvability and leniency for a game level)
- Relative performance (measure of a game-playing agent's competency in reaching a game's main goal or goals relative to other individuals in the population in coevolutionary setups)
- Multi-objective (indicates that a study employs multi-objective optimization rather than naïve weighted sum of multiple fitness contributions)
- Guidance (measure of an agent's participation in actions or behaviors that do not directly contribute to the agent's main goal or goals, in order to promote or dissuade from taking that specific action or behavior)
- Novelty (meta-information measuring how diverse an ANN is compared to other individuals, typically through a distance function)
- Interactive (fitness is evaluated interactively by a human)

Table 4.4. Fitness measure contributions distribution.

Fitness measure	N° of studies	%
In-game performance	96	81.36
Off-game performance	6	5.08
Relative performance	8	6.78
Multi-objective	11	9.32
Guidance	22	18.64
Novelty	9	7.63
Interactive	2	1.69

Each study can be categorized into multiple categories based on its content. As a result, the total percentage across all categories may exceed 100% because some studies contribute to multiple categories.

Agents playing games are featured in the majority of studies under review, as previously stated. Among agents playing games, the utility or fitness function almost always contains some measure of the agent's performance in the context of each individual game's goal. In such

uniformity, it is still possible to extract meaningful information by taking a deeper look at fitness function design in the examined work and analyzing the outliers.

Fitness functions of agents aiming for maximum performance in a game domain can pursue such goal differently based on the game's design. When the game domain features stochastic elements, fitness is typically evaluated over multiple episodes, and the results for each episode are then averaged to help smooth out the influence of random events. In most simple arcade games, such as many among those available in the ALE, the goal is to achieve a high score and an agent can only achieve so by playing the game correctly. That is the case when a game's score can only increase. What happens when a game has negative rewards for some actions? For instance, fighting games typically feature two opposing characters battling with the goal of setting their opponent's health value to 0 before they do. Therefore, it might seem intuitive to reward dealing damage while "punishing" (giving a negative reward) taking damage, as taking too much damage might cause the agent to lose the match. The issue with this approach is that an agent dealing an amount of damage and taking a comparable amount of damage back will have a numerical fitness value close to an agent who didn't deal or receive any damage, while having completely different behaviors. This can be extended to many other genres with direct confrontation and negative rewards (many of which can be found in this review, like Pong or units in RTS games like StarCraft).

The solution to this kind of issue leads to an interesting aspect of fitness function design. By adjusting what contributes (and how much it weighs in the calculation) it is possible to encourage or discourage certain actions, influencing an agent's behavior during evolution. This equates to narrowing the search space away from undesired deceptive local optima (e.g. inaction), requiring less generations to reach more desirable behaviors, but requires human intervention and domain knowledge. Introducing this kind of guidance in reward shaping might conflict with open-endedness of evolution and limit exploration.

Many studies under review employ this kind of reward shaping through action encouraging (or discouraging). In the Flappy Bird domain, where agents can perform a single "jump" action to go through pipes (a pair of obstacles with a traversable gap between them) without touching the edges and are subject to gravity, [63] attempts a minimal evolution strategy and achieves so after about 20 generations of a population of 30 individuals, rewarding agents closer to the gap at failure in the fitness evaluation. Unfortunately, unlike other testbeds such as the ALE, studies in the Flappy Bird domain under review use different implementations of the game, so it's difficult to make a direct comparison. Still, the size of the population and the number of generations are definitely on the low end.

In the SMB domain, the goal for agents is to complete levels. Levels typically develop horizontally, and the player character is prevented from going back once the camera scrolls over a specific level section. Hence, fitness is a function of the distance traveled from the beginning of the level, which can be normalized by the level's size. Coevolution of agents and levels in SMB found in [64] introduces a light penalty for jumps after experimental results showed that the evolution quickly got stuck into a local fitness maximum by simply moving right and jumping constantly, never learning how to jump in order to avoid enemies and gaps.

The experimental setup in [65], where SMB agents are evolved with a phased searching powered NEAT algorithm to achieve low complexity ANN structures for transfer learning purposes, define different guidance combinations for agent fitness. One setup measures distance traveled minus elapsed time, favoring speed. Another one adds rewards for coin collection and an additional one favors points scored by enemy elimination.

[66] in the Pong domain adopts a multi-phase fitness evaluation, measuring the paddle hits, goals for and against and static frames (to punish inaction). This study discusses the nature of

this fitness function's guidance as one of the reasons why the agents failed to successfully learn offensive scoring behavior, along with a small population size.

The study in [44] also reports struggling with evolving agents in the quite complex and sparse (reward-wise) testbed of Super Castlevania IV. Super Castlevania IV is a 1991 side-scrolling action platformer game for the SNES. While it shares similarities with the SMB domain, a few key changes in how the game plays translate directly into additional challenges for the agent. Levels can scroll back and forth, the world structure is interconnected rooms, which means that a room's exit might not simply be straight to the right. The agent has a richer action space with sometimes complex action sequences, being able to crouch, walk on stairs, interact with doors when in specific level positions, swing from specific elements during jumps by hitting them and holding the attack button. The agent in [44] guides the evolutionary process by rewarding score changes (always positive, can be earned by eliminating enemies and picking up items after breaking down objects), x-position changes (promote movement), weapon level increases, and punishing health loss. Level completion is not explicitly rewarded, although the game provides a score bonus proportional to the remaining time. Experiments with this setup show that agents aren't able to learn how to open doors in the first sections of the game, although on limited computational power.

Studies in the EvoMan domain [67][68][69] evaluate fitness as a function of player and enemy health, as agents try to take over opponents in one-on-one duels. [68] adds play time to the equation, promoting a more aggressive behavior.

Guidance in fitness function design can be also found in [70], where agents playing the two-button fighting game DropFeet (a clone Unity implementation of the fighting game Divekick) are rewarded for action variety and punished for inaction.

Agents evolved for providing players with a fair but challenging experience in a custom-built Unity racing game described in [71] have their fitness evaluated over their distance from the finish line (main objective), the total elapsed time, and the time correctly spent between a lane's boundaries (guidance). Experiments also show that adjusting the weight of these elements in fitness calculation can lead to diverse behaviors. A similar setup can be found in [72], where the goal is to develop believable racing game opponents. The fitness function rewards the number of completed laps and current track piece the car is in (the further the better) while punishing the number of collisions with walls.

The agents used to procedurally generate enemy waves in a custom-built tower defense game are evaluated on their in-game performance as well as severely punished for exceeding their unit-spawning budget, another form of guidance [32].

The game EvoCommander, introduced in [73], provides players with the means to configure their own fitness function by manually weighing different in-game performance parameters of their robot "brains", such as movement or combat goals. The game features actively taking part in the evolutionary process with the goal of evolving agents that can best their opponents in battles as a novel game design mechanic.

The procedurally generated endless runner games featured in [74] evolve agents by evaluating their fitness with the following goals: minimizing restarts (which happens at level failure, therefore constituting the main objective) and number of actions (to discourage unnecessary or random command inputs).

Another observable pattern in fitness function design among studies under review is the usage of multi-objective evolution. In multi-objective evolution there is more than one fitness function, and these multiple functions may have conflicting goals, where progressively achieving a high fitness in one function can degrade fitness in another function. The aim is to

optimize for the Pareto Front, the solution subspace where individuals can not improve their fitness without lowering it in other objective functions. More information can be found in Chapter II.

This differs from adding multiple objectives to a single fitness function, as described previously, because it is possible to keep track of what kind of behavior is responsible for an individual's fitness. At the same time, it introduces a new problem, which is the necessity for sorting genomes over different parameters.

In [75] MM-NEAT (a NEAT variant supporting multi-objective evolution through Non-dominated Sorting Algorithm II, or NSGA-II) is used to evolve a bot playing the FPS game Unreal Tournament 2004, aiming to develop desirable teammate behaviors for a better player experience. The fitness function optimizes the team's score and damage dealt, minimizing damage taken at the same time. Although the main objective is team score, the additional non-contradictory objectives provided smoother gradients.

Two-objective fitness sorted with NSGA-II can also be found in studies addressing combat unit AI in the RTS game StarCraft II [76][77]. The fitness functions similarly evaluate damage taken and dealt as separate objectives.

The authors in [58] report that having the elapsed time along game score as objective for Tetris agents helped the early steps of the evolutionary process, where agents haven't learned yet how to clear lines for scoring. The same fitness setup is found in the previous study [50].

[20] employs several fitness measures to evolve a FightingICE agent competing in the Fighting Game AI Competition. These make use of both episode-related information as well as information on the history of the agent's episodes, as the different objective functions track the agent's damage dealt and taken in a fight, the number of close combat attacks (to discourage ranged only strategies), and the win ratio against a specific opponent. Because of the nature of fighting games, where offensive actions expose players to a few frames of vulnerability, the damage taken and dealt objectives are partially conflicting.

Multi-objective evolution can also be found in studies in the procedural content generation category. [59] and [31] focus on generating large-scale levels for SMB and Zelda. The multi-objective approach found in these studies differs from the previous ones, as it makes use of Multi-dimensional Archive of Phenotypic Elites (MAP-Elites). The idea is to subdivide the individuals in "bins" based on different behaviors or features (the multi-objective aspect), then selecting the highest fitness individuals from the various bins, facilitating exploration in the evolutionary process. Specifically, the mentioned studies measured the percentage of non-standard tiles (question blocks, pipes, enemies etc.), percentage of tiles on which the Mario character can stand, and the average leniency across all tiles in the SMB domain, whereas in the Zelda domain wall and water tile percentage was measured, with the addition of the number of reachable rooms. Fitness within different bins for both domains is the length of the shortest path the agent must take to complete the level, with 0 meaning the level is unsolvable.

In [32], where an agent is evolved with a Quality Diversity algorithm to build structures in the Voxelbuild framework, both a fitness function (quality) and a behavior characterization (diversity) have been defined. Fitness is based on the procedurally generated content features. More specifically it takes into account the net number of blocks used (placed minus removed) multiplied for the structure's height in blocks, rewarding tall structures more. Structures are represented by a vector of binary values, where the index maps a block to coordinates in the world and the value indicates whether the block exists.

The Wuji game-testing framework featured in [60] is another example of two-objective setup combining game winning goals and a measure of game states exploration (how many different

game states are triggered by a given sequence of actions). This is different from novelty, as it is related to episode performance and is not meta-information on genotypes or phenotypes.

[8] and [78] propose multi-task and multi-objective optimization in the Atari testbed respectively, to evolve general game-playing agents capable of playing multiple Atari games. [78] specifically, which follows up [8], provides comparison between single and multiple objective setups. The single fitness function for Atari games evaluates the game-playing performance in the form of score. The multi-objective setup adds the elapsed time as a second measure of fitness. Experiments show that this approach is able to evolve networks outperforming both single task and single objective networks.

A coevolutionary solution leveraging relative performance as a measure of agent fitness is described in [18], where competitive coevolution and traditional neuroevolution are compared in the RTS game "Starcraft: Brood War". The fitness function for NE evaluates rewards a positive ratio of health over enemy health and punishes elapsed time, discouraging inaction. In competitive coevolution, fitness is relative to other individuals. Similarly, in [61] various coevolutionary methods are compared in the Atari testbed, evaluating fitness relatively to other individuals in the population.

Aside from game-playing agents, a few different definitions of utility can be found. In [79] two populations of individuals are coevolved to imitate high performance RL pre-trained static deep RNN agents. In this GAN setup, generators are randomly matched with discriminators and their fitness is evaluated based on how incorrect the paired discriminator assessment is and discriminators have their fitness evaluated on how correct their assessment is.

Studies concerning the NEATEST automated playtesting framework for Scratch [28][29][80] use linear combinations of code coverage indicators (how many statements of a given program can be reached by the ANN input test generator).

A very important alternative (or valuable aid) to performance-based fitness is novelty, or diversity. Novelty is a form of meta-information that does not evaluate an individual based on its task performance. Instead, it serves as an indication of how different each genotype or phenotype representation of a solution differs from other individuals in the same population, encouraging exploration. This usually brings up the problem of defining a distance function. The study in [81] employs the novelty metric in SMB procedural level generation, using a compression distance function that measures the size difference between the compressed representation of two levels together and separately. [47] also compares the Euclidean distance between an individual and its neighbors in latent space to calculate novelty scores in Minecraft buildings. [48] extends the method in [81], moving from the 2D grid systems used in SMB to the three-dimensional world of Minecraft and using the novelty search evolutionary method in a hierarchical multi-level generator. Fitness for generators promotes novelty (diverse generators), intra-generator novelty (diversity in levels generated by an individual generator), and feasibility (whether a level can be solved).

Another case of meta-information in fitness function design can be found in [38]. The context in this study is to compare neuroevolution and gradient descent in preference learning, evolving an agent to accurately predict player arousal from annotated player footage. The loss function, typical of binary classification tasks, becomes the fitness function used in the evolutionary process. Training loss is also featured in [21], where cultural algorithms are used to optimize ANN topologies of pre-trained (through reinforcement learning) DNN agents in the FightingICE framework.

A unique blend of in-game performance and meta-information in fitness function design can be found in [27]. The study addresses the problem of game balance through neuroevolution of AI

agents, using the game Ms PacMan as a testbed. Fitness evaluation of individuals is the sum of the distance between the desired and average score of the agent (over multiple games) plus the distance between the observed and desired standard deviation. Experiments aiming at maximum performance, specific average score and specific standard deviation are performed, proving this approach as a useful tool for adjusting AI skill levels.

High fitness individuals can also be selected manually, which can help navigating domains where there is not a clearly measurable goal. This is the case for interactive evolutionary processes, as described in [82] and [30]. In [30], the user influences its own experience in the Infinite Art Gallery, a 3D world of procedurally generated 2D and 3D artwork. Navigating the gallery by interacting with artwork will take the player to new rooms where offspring of the chosen artwork are featured. Interactive evolution can be cumbersome for users, especially considering the iterative nature of neuroevolution. [82] attempts to solve this issue by distributing the workload in a collaborative interactive setup, where multiple users contribute evolving partial solutions (complex maze navigation in Minecraft).

In summary, the straightforward measure of fitness for game-playing agents is the in-game performance of the agents in terms of the game's main objective. Similarly, agents with specific off-game goals (e.g. code coverage in automated playtesting, training loss in classification task, imitation behavior confidence etc.) are evaluated on their performance on the given task. Coevolutionary solutions of agents sharing a goal often measure fitness relative to other individuals in the same population. This makes it harder to evaluate agents quantitatively, but the issue can be overcome with approximation through external evaluators, as mentioned in [61]. In procedural content generation, fitness is a measure of hand-designed content features, often including some feasibility measure for game levels.

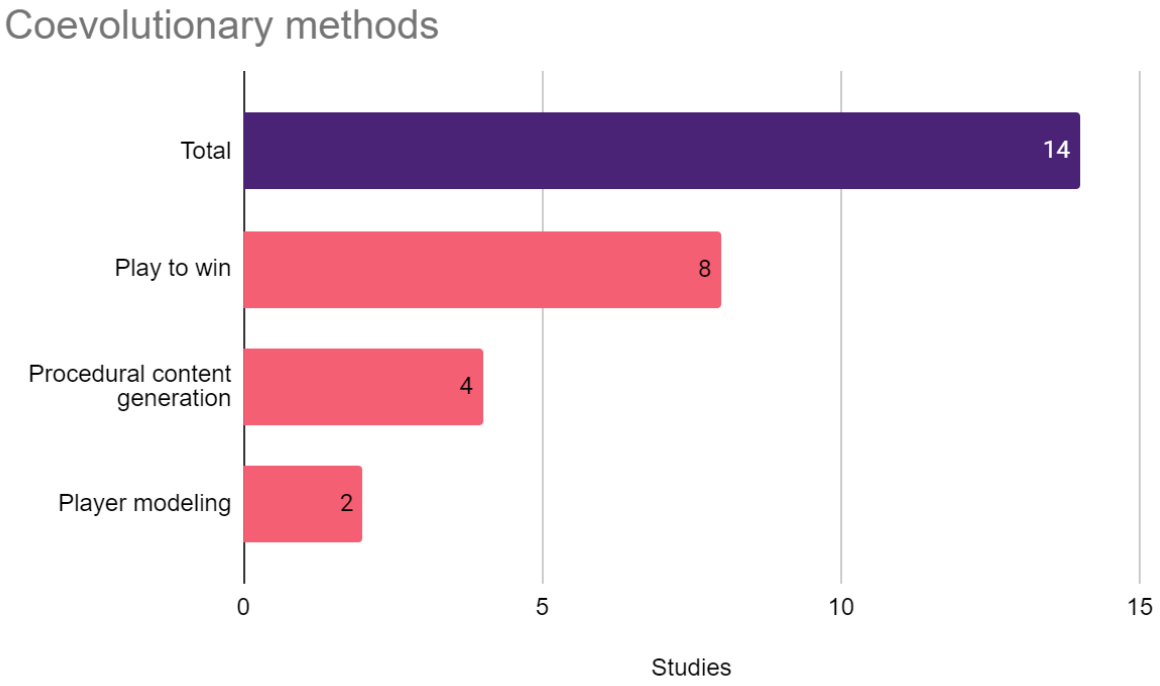

Figure 4.1: coevolutionary methods distribution in categories.

Fitness measures can also be fine-tuned to include additional information in reward shaping, aiming to guide the evolutionary process by encouraging or discouraging specific actions and

narrowing down the search space. The additional information can be included as a weighted sum in the fitness function or there can be multiple fitness functions. This second approach, multi-objective evolution, introduces computational overhead for sorting solutions based on the different fitness functions, but can promote diversity and highlights what contributed to a high fitness individual in a clearer way.

Meta-information can also be used to influence the evolutionary process, as it is often the case with novelty. Measuring diversity in solution representations has been shown to benefit a few studies in the Atari 2600 testbed [83][84], as well as many procedural content generation neuroevolutionary techniques, where promoting exploration is particularly important.

Finally, it is also possible to interactively pick high fitness individuals through human interaction in the evolution process. While cumbersome, this approach can help solving problems with unclearly definable goals.

4.4 Evolutionary Algorithms

The last technical implementation aspect of neuroevolutionary solutions in games under review is the evolutionary algorithm choice. There are three main branches that encompass most of the solutions found in the work pool: Evolutionary Strategies, Genetic Algorithms and Neuroevolution of Augmenting Topologies and its extensions. An overview of these algorithms can be found in Chapter II. The outliers which do not fall under this classification will be discussed at the end of the section.

Table 4.5. EA family distribution.

Input type	N° of studies	%
ES	18	15.25
GA	31	26.27
NEAT	66	55.93
Other	10	8.47

Each study can be categorized into multiple categories based on its content. As a result, the total percentage across all categories may exceed 100% because some studies contribute to multiple categories.

4.4.1 Evolutionary Strategies

It can be observed that ESs are a popular choice in the visual input domain, being featured in almost half the studies, and frames being the input to 12 out of 18 solutions adopting ES as the EA of choice.

[45] is seminal work which is often cited in other studies in the Atari 2600 testbed cluster, where an ES approach is applied to RL demonstrating the scalability of this kind of solution thanks to parallelization and its invariance to frame-skipping (observations have a lower frequency compared to the simulation environment).

[83] extends [45] with Novelty Search and Quality Diversity algorithms to address the local optima avoidance problem in Atari games. Among the proposed methods, the quality diversity

hybrid NSRA-ES improves over [45] in escaping local optima in several Atari games by dynamically promoting novelty when necessary (to avoid unnecessary exploration pressure in less deceptive domains that do not need it), while maintaining the scalability and sample efficiency features that ES provide.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is featured in a good number of these studies. [85] compares a CMA-ES evolved ANN which is fed latent space dimensionally reduced frames in 50 Atari 2600 games against various baselines, including [45] and RL solutions. Experiments prove this approach to be more sample efficient (less than 10 million frames against billions in RL), computation efficient, and to converge faster.

[86] takes a different approach to the same testbed, leveraging a self-attention module evolved through CMA-ES in place of an autoencoder. The compressed input is then fed to a NEAT agent, which evolves alongside the self-attention module in a hybrid solution. The benefit to this is that feature extraction does not require prior knowledge on the domain, and such dimensionality reduction is needed for NEAT to perform well on larger networks. Agents evolved with this method do not achieve record breaking performance, experiments on a subset of 4 Atari games showed that this method was able to achieve comparable performance (fitness) to the baseline with a lower number of parameters. On the other hand, the coevolution of two populations makes the process time intensive.

The study [87] tries to extend CMA-ES by introducing a restart-based rank-1 evolution strategy (R-R1-ES). This simplified CMA-ES variant method differs from other ESs in its implementation and is able to adapt distribution mean, mutation strength and search direction. It also leverages a restart procedure to help escape local optima. Experiments show that this variant outperforms the baseline ES methods, [45] and [83] among them.

[88] aims at evolving an Atari 2600 ANN agent decoupling feature extraction from visual input and decision making, a common dimensionality reduction theme in composite neuroevolutionary solutions based on visual learning. The feature extraction leverages vector quantization (Increasing Dictionary Vector Quantization, or IDVQ). The study focuses on using the smallest possible ANN topology, achieving results above random policy and below human scores while being two orders of magnitude smaller than state-of-the-art solutions.

[89] introduces GENE, an indirect encoding for DNNs that leverages a geometric approach (low dimensional Euclidean space projection) on a fixed-size fully connected ANN and using distance functions for genotype to phenotype mapping. Experiments aim at demonstrating that evolution strategies like Separable Natural Evolution Strategy (SNES) and Exponential Natural Evolution Strategy (XNES) can benefit from a smaller search space. Results showed that it is possible to achieve scores comparable to direct encoding with more robustness (lower standard deviation) and computational costs on the update step can be significantly better for large genomes in XNES.

Studies in [8] and the following [78] address transfer learning in the Atari 2600 domain with Multi-Task Evolution Strategy (MTES) and Multi-Task Multi-Objective Evolution Strategy (MTMO-ES) for general Atari game playing, extending [45].

MTES adopts multi-task learning to improve generalization capabilities for DNNs. The aim is to extract shared features and take a new approach to transfer learning. Experiments show that DNNs can learn up to four Atari games simultaneously and the Pareto front convergence highlights shared features.

MTMO-ES extends [83] and adds multi-objective optimization. Results from experiments run on a single DNN on two tasks (Atari games) and with two objectives (score and play time)

prove that a DNN evolved with this method has excellent performance over its single-task single-objective equivalents, while expected to be more adapt to general game playing.

Outside the Atari 2600 domain, ESs are used for coevolution of opposing factions in the Hnefatafl asymmetrical board game [24]. A value network and trainable filters are used in order to improve simulation time and population quality through tournament selection. Except for the first round, the highest fitness individuals are selected for the tournament.

In [53], ensemble learners are used in Leduc poker playing (a simpler poker variant used in research). The study proposes three solutions: an attention-based solver, a gradient-based solver, and an evolution-based solver (neuroevolution-based centralized training, or ES-NCT). The gradient-based and attention-based solvers are base solvers for ES-NCT. Results from experiments comparing ES-NCT to its base-solvers and baseline algorithms proved that ES-NCT has superior performance and adaptability.

CMA-ES for decision-making networks over dimensionally reduced visual input also proved to be valid in the VizDoom domain in [17]. An autoencoder trained with backpropagation compresses the visual input.

Self-attention and CMA-ES finds success in the CarRacing (2D top-down view racing game) and DoomTakeCover (a specific VizDoom setup in which the agent has to survive as long as possible by dodging enemy fire) domains. The authors in [56] draw a parallel between indirect encoding and self-attention in their ability to produce comparatively small networks for high complexity visual tasks. Experiments show better performance (fitness) compared to the baseline (including [55]) while being unaffected by perturbations to task irrelevant aspects (color or texture change, addition of elements not occluding critical information and similar alteration of the visual frame).

Coevolution of agents and environments is used to achieve complexity and diversity in procedurally generated levels in a custom 2D game in [34]. Agent evolution uses Canonical Evolutionary Strategy (CES). The environment was represented by a Neural Cellular Automata (NCA) and evolved with an ES. Fitness in this case was the Euclidean distance between the output and a real map. During the coevolution process, environments are eliminated based on a fixed criterion (map feature density) and agents are tested on the remaining environments. Experiments compared directly evolved agents with coevolved agents on existing maps, highlighting better performance (fitness) for coevolved agents, while giving no insight on the quality or diversity of procedurally generated maps.

GAs are also the EA of choice in studies that use FPGA hardware acceleration [90][13]. In [13] a DNN is evolved with neuroevolution on a multi-FPGA framework (the Versatile Tensor Accelerator, or VTA). Experimental results show that learning times scale almost linearly and are only bound to the population size. Furthermore, training over VTA happens at a fraction of the power consumption.

4.4.2 Genetic Algorithms

Genetic algorithms are a bit more popular than ESs and see more variety in both goals and input representations, including difficulty control, behavior imitation and procedural content generation on top of the predominant optimal game-playing domain.

In [91] a simple GA is tested against gradient descent methods and the ES from [45] in the Atari 2600 environment. Results from experiments show good performance (fitness) on many games, potentially constrained by the limited computational power used. The authors extended some evolution runs past the baseline amount of training frames and achieved increasingly better

performance without converging. GA was also found to detect high performance individuals in less generations compared to the other methods. The authors observe that GA outperforms ES because the random parameter perturbations produced diverse enough results saving valuable time compared to ES calculating the updated parameter vector, while the GA also does require the overhead due to virtual batch normalization. A Novelty Search variant is used to avoid local optima, a similar practice that was also found among ES-based studies in the previous section.

A custom real-time first-person fighting game is used to test neuroevolution for dynamic difficulty adjustment in [92]. A GA evolves the network's weights, featuring two-point crossover. In addition to this setup, catalogs are used to store snapshots of different neural networks during training, allowing difficulty tuning by loading previous catalogs. No experimental results were shown.

[71] also attempts at evolving agents to reach a desired level of skill in order to provide an adequate challenge for players. The testbed for the study is a custom-built racing game in the Unity game engine. Agents were evolved with a GA featuring a population of 25 and elitist selection. Experiments show that it is possible to obtain diverse behaviors by adjusting weights within the fitness function definition.

In [79] two populations of individuals are coevolved to imitate high performance RL pre-trained static deep RNN agents activated by ReLU. In this GAN setup, generators are randomly matched with discriminators for fitness evaluation. Evolution is based on a simple GA with no crossover or speciation methods. Experiments proved successful on various control tasks, the game LunarLander being one of them.

The study in [55] attempts to evolve modular networks for visual input processing (autoencoder), memory and decision-making end-to-end rather than separately using a simple GA. One of the advantages of GA is that it natively supports discrete representations, which are difficult for gradient-descent, making it possible to evolve a discrete (through step function) autoencoder. Experiments run on the CarRacing testbed show fitness very close to the world models baseline, while outperforming traditional RL methods on a comparatively small population of 200.

[52] is another case of neuroevolution of modular networks through GA, this time in the Poker domain. The agent is trained against a random subset of a set of bots with diverse playstyles, with no subset made of a single type of bot, combining curriculum learning with neuroevolution.

4.4.3 NEAT and its extensions

The Neuroevolution of Augmenting Topologies, a neuroevolutionary technique proposed by Stanley and Miikkulainen in 2002 [5], is by far the most common class of algorithms (along with numerous extensions) adopted by studies in the work pool. Its main features are the ability to simultaneously evolve ANN weights and topologies, the usage of historical markers to avoid redundant structures, and innovation protection through speciation to maintain diversity. NEAT is introduced with more details in Chapter II.

HyperNEAT is a popular NEAT variant that uses generative (indirect) encoding through CPPNs. This allows HyperNEAT to indirectly encode large networks that contain symmetries or repeating patterns without the computational cost that NEAT encounters when evolving high-dimensional networks. At the same time, the added complexity of HyperNEAT can make it less efficient than NEAT in low-dimensional spaces, as discussed in [49]. In this study neuroevolution, CMA-ES, NEAT and HyperNEAT are compared on the Atari testbed on various

dimensional input sizes (different color representations for raw pixels) and game state input. Results from experiments (and previous studies) showed that these solutions perform best on low-dimensional inputs (game state), NEAT outperforming HyperNEAT except on 8-color pixel representation.

[50] focuses on comparing direct and indirect encoding for learning Tetris from raw screen input and game state features. NEAT and HyperNEAT are used for evolution, and the fitness function is multi-objective, as mentioned in the previous section, evaluating score and time steps sorted through NGSA-II. Results from experiments show that NEAT agents were not able to play on raw visual input, while scoring better on game state features, surpassing HyperNEAT's initial advantage over generations, underlining the theme of game state features producing higher fitness individuals compared to visual inputs, and indirect encoding is a better choice because of the high dimensionality of raw pixel input.

The study in [69] compares a GA, NEAT and RL in the EvoMan framework, evaluating the impact of noise perturbation in the environment (enemy position randomization). Results from experiments show that EAs perform worse than Proximal Policy Optimization (PPO) under noise perturbation, but comparably well without noise, while being much more efficient in training time. This provides further proof of the sample efficiency of evolutionary algorithms already encountered in previous sections.

[20] features another multi-objective NEAT variation inspired by the hypervolume selection combined with non-dominated sorting of SMS-EMOA [93]. nNEAT uses NEAT's representation and variation, while the SMS-EMOA component covers fitness functions, population, parent and survivor selection for multi-objective optimization. This work introduces two sorting methods for nNEAT: a Quality Diversity measure and iterative R2. Results from experiments in the FightingICE framework showed that the nNEAT can evolve agents outperforming most rule based and ANN opponents in the baseline with a positive win ratio.

[22] extends the nNEAT agent from [20] with two modifications. One variant changed the type and amount of input and output neurons (to reduce complexity), while the second takes advantage of input preprocessing. Results from experiments prove these solutions as a valid approach for this domain, although it did not compete in the Fighting Game AI Competition at the time the study was published.

MM-NEAT, another NEAT variant supporting multi-objective evolution through, or NSGA-II, is used for evolving bot behaviors in the FPS game Unreal Tournament 2004, as mentioned in the previous section [75]. It is also part of CPPN2GAN, a combined solution made of a pre-trained GAN and a CPPN evolved by MM-NEAT for the procedural generation of large-scale 2D game levels [59][31].

Similarly to the previous sections, a vision-based game-playing agent solution based on an autoencoder for dimensionality reduction can be found in combination with NEAT. [46] introduces Autoencoder-Augmented NEAT (AE-NEAT) for this purpose. This method trains both the autoencoder and the decision-making network (policy) in unsupervised fashion. The Atari annotated RAM representations are used to evaluate the autoencoders' feature extraction capabilities. Experiments on Atari 2600 games show competitive or exceeding performance in a small subset of games compared to human levels and other evolutionary algorithms, but inferior to state-of-the-art RL methods.

[94] proposes a different angle by enhancing Policy Gradient Search (PGS) with NEAT. The hybrid solution separates feature learning (from RAM states) and policy learning (which in

game-playing agents corresponds to the decision-making network) for Atari 2600 optimal game playing. Experimental results show that this solution performs comparatively to the baseline (various gradient-descent RL methods and base NEAT), excelling in the Seaquest game, while often being more sample-efficient.

Transfer Learning for intelligent gameplay (in the SMB domain) is attempted by aiding NEAT with phased searching, a method for finding ANNs with smaller topologies, in [65]. The idea is to learn a source gameplay task and use the highest fitness individual for the initialization of the target gameplay task. The phased searching algorithm relies on setting the removal probability for genes based on a gene's rank (weight values for links, number of links for neurons). Experiments on transfer learning comparing this approach to other phased searching strategies prove that the solution outperforms the baseline methods and is able to control the ANN complexity.

[95] approaches general video game playing by combining NEAT with Rolling Horizon Evolutionary Algorithm (RHEA) in the General Video Game AI (GVGAI) framework. RHEA is a statistical forward planning method that relies on forward models to simulate the future effect of actions in the current game state. A very popular algorithm that belongs to this class is the Monte Carlo Tree Search (MCTS). RHEA's individual representation encodes a sequence of actions. Experiments compare baseline Rolling Horizon Neuroevolution of Augmenting Topologies (rhNEAT) with various extensions and alterations of itself (leveraging NEAT's flexibility and modularity) and baseline MCTS and RHEA in several GVGAI games. The extensions include NEAT with population carry, speciation and both population carry and speciation. Results show that both additions contribute to improvements in game score and win rate, but overall, the solution performs worse than baseline algorithms with a few exceptions that warrant further investigation in this method.

Zhang et al. propose two reverse encoding tree search methods for NEAT with adjacency matrix edge encoding as representation [96]. The proposed methods leverage binary search (Bi-NEAT) and golden-section search (GS-NEAT) and are tested against baseline NEAT and feature-selection NEAT (FS-NEAT). Experiments on control tasks and the game Lunar Lander show that these new methods can achieve the same fitness in fewer generations compared to the baseline, with Bi-NEAT performing better than GS-NEAT in tasks without noise perturbation.

[38] introduces RankNEAT, a simplified NEAT algorithm which does not add new nodes to the ANN topology. The study compares this algorithm to the baseline RankNet, a stochastic gradient-descent preference learning algorithm. The ANNs start fully connected and the evolution process can only remove edges, to avoid overfitting. The solution compares favorably to SGD in the domain of preference learning of player arousal from gameplay footage with the aid of pretrained computer vision models (a Vision Transformer).

NEAT also serves as the foundational algorithm in studies utilizing interactive evolution [82][30].

Further information concerning NEAT and its extensions can be found in the systematic literature review by Papavasileiou et al. which includes applications outside the games domain [136].

Table 4.6. Hybrid and composite methods.

Studies	Hybrid or composite method
[85]	AE + ES
[55]	AE + GA
[47] [46]	AE + NEAT
[51]	Context + Skill
[83]	ES + Quality Diversity
[52]	GA + Curriculum Learning
[74]	GA + Depth-First Search
[23]	GA + RL
[100]	Hierarchic Memetic Strategy
[88]	IVDQ + ES
[60] [101]	Multi-objective EA + RL
[31] [59] [102]	NEAT + GANs
[81] [48]	NEAT + Novelty Search
[94]	NEAT + PGS
[103] [9]	NEAT + RL
[86]	NEAT + Self-attention
[28]	NEAT + SGD
[99]	TWEANN + Ensemble Learners

4.4.4 Other EAs

[21] introduces CATNeuro, a neural architecture search that combines graph evolution as described in NEAT with Cultural Algorithms, an evolutionary algorithm class inspired by cultural evolution principles such as knowledge sharing and social learning that optimizes search in a population. Two main methods are compared: the competitive weighted majority distribution, and the cooperative stag-hunt distribution. Experiments in the FightingICE framework prove this as a valid approach, although the resulting agent was limited by the RL model used to generate the training data.

[97] tackles open-ended learning with PINSKY, or POET-Insipred Neuroevolutionary System for Kreativity. POET is Paired Open-Ended Trailblazer, an open-ended agent-environment coevolutionary algorithm, which adopts transfer learning to optimize agents. PINSKY extends its application to the GVGAI framework. The analysis focuses on evaluating the importance of transfer, which is proven by experiments in developing optimal agents, while also underlining the need for a minimal criterion in environment generation.

The work in [98] proposes an alternative Topology and Weight Evolving Artificial Neural Network (TWEANN) algorithm, Artificial Life Form (ALF). ALF uses direct encoding and has three prominent features: semantic and structural speciation, dynamic adaptation of population and fitness-based genetic operators. ANNs receive sensory input and map their outputs to actions. Experiments comparing ALF with NEAT and run on three SMB levels show that ALF has better performances in runtime and ANN size.

Another TWEANN variation can be found in [99]. In this study, the evolution of ensemble learners is proposed to restrict the search space. Ensemble learners consist of a combination of predictive models used to achieve better robustness and accuracy compared to the set of base models they are composed of [138]. A divide and conquer approach is used to overcome limitations in evolution time, non-deterministic environments, and overfitting. Output distribution and bootstrap aggregating are the selected ensemble learning methods used in experiments on SMB, Snake, and a control theory problem (double pole balancing). Results show the output distribution approach outperforming the ALF [98] baseline in score and learning time.

5. Play to Win

In this first and largest category, the goal for neuroevolution-based solutions is to provide high-performance game-playing agents. In the previous chapter, it has been observed that this translates to an ANN topology where the inputs represent some kind of encoding of game observations, typically visual frames or game state features, and the output represents the selected action for the agent.

Clustering for this macro-category aims at grouping studies based on their testbeds, in order to evaluate the shared characteristics of various problem spaces and understanding different use cases for neuroevolution in games.

There are three main types of clusters: game-based, emulator-based, and genre-based. Games with a significant number of studies specifically dedicated to them are grouped together for comparison purposes. It can be noted that some single game labels represent games that can be found in studies concerning framework-based solutions, specifically those based on emulators. Examples of this are Tetris and SMB (which are available on the NES), or Pong (which can be found in the ALE). However, it is relevant to consider them as separate groups because studies in emulator-based environments present solutions that attempt to achieve various degrees of generality in their game-playing agents, leveraging the availability of multiple games within the same framework, while game-specific studies focus on different aspects.

A large body of work uses Atari 2600 games as testbed for a variety of algorithmic solutions. The Arcade Learning Environment (ALE) is a popular framework designed to easily allow for the development of AI agents that can achieve general video game playing through emulation. ALE is also featured as part of OpenAI Gym, a collection of RL environments [104]. Neuroevolution solutions in this domain do not achieve record-breaking performance fitness wise but prove to be more efficient in training [83][88][86][94][45][85] and produce simpler topologies [42]. Most of these networks process visual input for generalization purposes, while a few leverage the RAM states provided by the ALE. [42] reviews the quality of RAM annotations on a per-game basis. ES are quite popular in this domain, because of the breakthrough work in [45] on which many studies have built and is often referenced as baseline and share the starting DNN. In many of these solutions neuroevolution serves as a cog in a larger modular setup and seems to be best suited to evolve decision making networks dealing with dimensionally reduced input via autoencoder [46][85] or other means [86][94]. Comparisons between various algorithms and input representations in [49] show that neuroevolution performs best on game state inputs in this domain, although studies focus on visual input for generality.

The other emulator-based environment is for NES and SNES games. While also available in OpenAI Gym, it's not as widely popular. Games for these systems have richer graphics (both resolution and color depth wise) and can feature deeper and more complex decision making compared to Atari 2600, making it more computationally expensive to research.

The most popular single game under review is Flappy Bird, which is a very simple game in both goals and gameplay. The player has access to a single action, has a single goal and a single failure condition. This simplicity might explain the popularity as a testbed. Although it is difficult to extract purely quantitative data because there is no standard implementation of the game for this purpose, most of the solutions successfully implement neuroevolution. [105] and [106] are shown to outperform RL agents in this testbed. 11 out of 15 solutions employ NEAT

and are able to evolve networks with minimal topologies, with the rest being GAs and a single implementation combining traditional neuroevolution with backpropagation. All the solutions in this space use game state features as ANN input nodes and measure fitness in terms of the game's main goal, which can be modeled either as score or distance traveled, as there aren't significant nuances differentiating them. [63] provides additional guidance in the fitness function, promoting faster convergence.

Table 5.1. Play to win studies distribution, grouped by testbed.

Studies	N° of studies	% Share	Testbed
[88] [90] [62] [87] [83] [8] [84] [86] [49] [61] [94] [78] [45] [137] [100] [91] [89] [42] [46] [85] [57]	21	24.70	ATARI
[24] [26] [25]	3	3.52	Board Games
[20] [21] [22]	3	3.52	FightingICE
[107] [51] [105] [106] [63] [108] [109] [110] [15] [111] [112] [113] [114] [115] [116]	15	17.64	Flappy Bird
[17] [9] [103] [56]	4	4.70	FPS
[52] [117] [23] [53]	4	4.70	IIGs
[67] [68] [69]	3	3.52	EvoMan
[44] [118] [43]	3	3.52	NES/SNES
[13] [66] [14] [119]	4	4.70	Pong
[55] [120] [22] [56]	4	4.70	Racing
[18] [76] [77]	3	3.52	RTS
[99] [121] [122] [64] [65] [98] [123]	7	8.23	SMB
[50] [58]	2	2.35	Tetris
[95] [124] [97] [125] [82] [126] [127] [128] [129] [130] [96]	11	12.94	Miscellaneous

Each study can be categorized into multiple categories based on its content. As a result, the total percentage across all categories may exceed 100% because some studies contribute to multiple categories.

5.1 Contributions to RQs

The following sections aim to extract relevant information towards answering each of the RQ defined in Chapter III.

5.1.1 RQ1

What are the most common neuroevolution techniques in the context of developing digital and non-digital games?

Based on the data extracted in Chapter IV, all these solutions share many common traits. Except for the chess board evaluator in [26] and the high-level goal selection ANN in [9], all the neuroevolutionary techniques used in play to win scenarios represent an agent directly playing a game. This translates to ANN structures featuring observations as inputs (in the form of raw visual frames, RAM states or game state extracted features) and producing action outputs (which can be controller actions, often found in emulator-based studies, higher level game related actions like attack or move for action or more generally dexterity-based games, or actions like raising or calling a bet in Poker etc.).

Visual frames are the input representation of choice for generalization purposes. They are most common in the Atari 2600 testbed: 16 out of 24 total visual frame representations concern studies in the Atari 2600 domain, which features 21 total studies. Starting networks tend to be DNNs with a similar correlation, reflecting a pattern established in the pivotal study [45]. The choice of ESs as EAs is also tied to this trend. Information-imperfect games and racing games agents found success in the usage of LSTM cells for exploiting opponents and learning tracks.

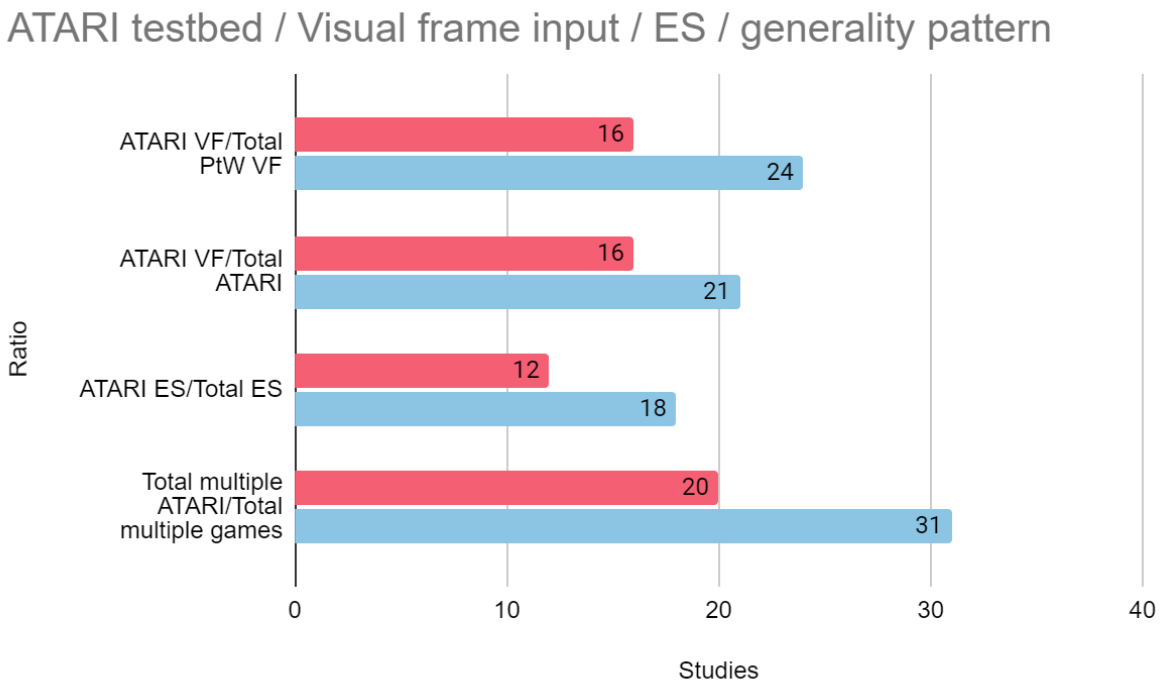

Figure 5.1: ATARI 2600 testbed provides major contributions to visual frame input representation, ES adoption and generality.

Most play to win agents evolve both network topology and weights through NEAT or extensions over game state extracted features, with significantly smaller starting networks (often without hidden layers).

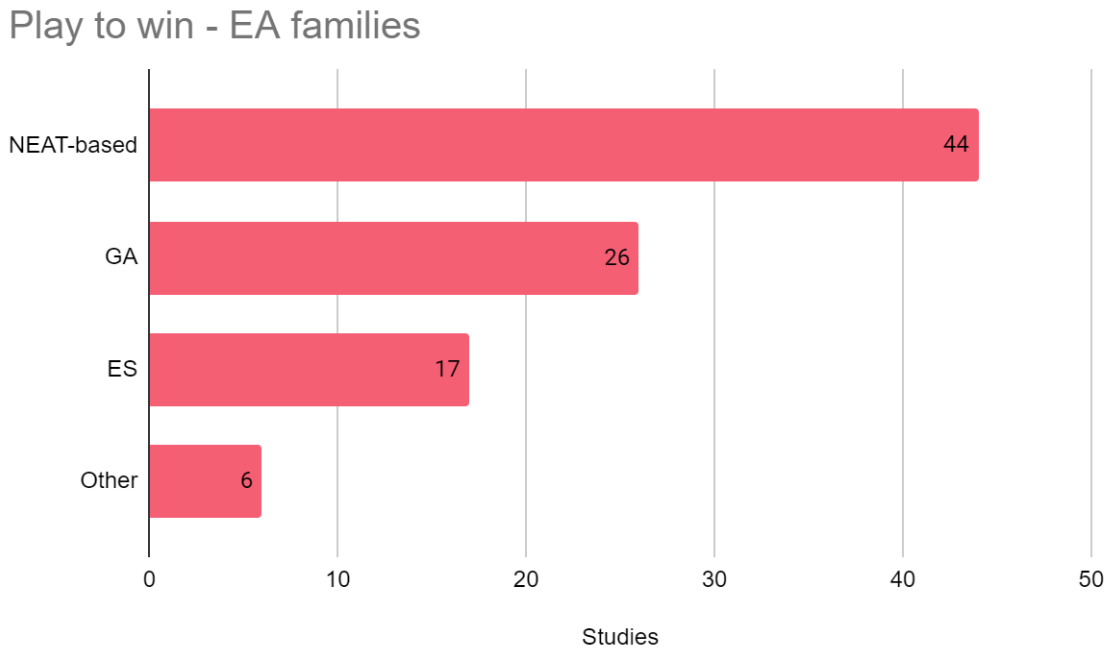

Figure 5.2: EA distribution in play to win category.

Fitness function design measures the agent's in-game performance for the evaluation step. This may vary depending on the specific game domain, but there are many shared traits in the studies under review. Games involving combat (fighter games, FPS games, RTS unit skirmishes etc.) involve an agent's health as well as their opponent's, games where the main goal is navigation based like Flappy Bird and SMB measure the distance traveled by the agent, gambling-related games (Poker, Blackjack) measure return of interest and win ratio.

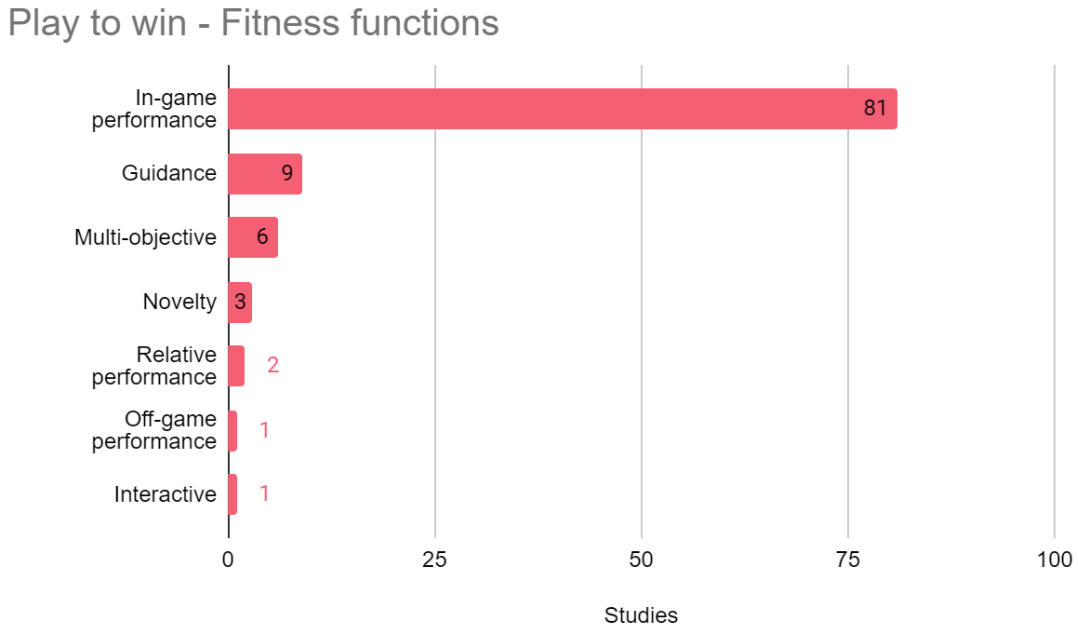

Figure 5.3: fitness function contributions distribution in play to win category.

Studies in several game domains find success in taking advantage of guidance in fitness function design to narrow down the search space, avoiding undesirable behavior like inaction, excessively frequent action without purpose, ranged attack abuse in fighting games.

In multi-objective fitness functions, NSGA-II is the most popular sorting algorithm, both for competing and non-competing goals.

5.1.2 RQ2

What are the potential use cases for neuroevolution in the context of developing digital and non-digital games?

Something that can be observed for all the games in this macro category, apart from board games, is that the games in this domain are of simple complexity. The in-game action space for agents, to be intended as effective actions that can be performed in the game context and not as controller inputs (e.g. Mario can walk, run, jump, crouch, and shoot but does so through the combination of controller inputs, which correspond to the ANN outputs) is limited to little more than movement actions. Games in the Atari 2600 domain have simple arcade designs bound by technical limitations of their time. Flappy Bird can only perform a single action. Pong paddles move along a single axis. Doom agents can move and shoot but their evaluations are limited to specifically designed test environments. Racing cars can steer and alter their speed through acceleration and braking. Poker and Blackjack have a limited action space, mostly interacting with bids. The FightingICE testbed provides the most complex environment in this sense, because the action space is quantitatively larger.

In the RTS games, which provide a much more complex and high-level action space, neuroevolution solutions in studies under review are limited to single unit control. Normally, RTS players are in charge of many more decisions than unit control. For example, the game Starcraft II featured in [76] and [127] allows players to do much more than fighting with units: players can build structures, gather resources, produce units, or research upgrades. The ANN agents only control combat units whose action space includes movement and attacking or healing (both modifications to the same variable, another unit's health).

5.1.3 RQ3

What are the costs and benefits of utilizing neuroevolution in the context of developing digital and non-digital games?

The large body of work provides a good overview on the flexibility and adaptability of neuroevolution-based solutions in developing competent agents. It's difficult to compare the various methods with each other because of the different testbeds and the variety of solutions, especially composite, hybrid, and modular ones with a lot of moving parts. The most popular testbed, which provides valuable comparisons is the ALE. Neuroevolution in this domain does not achieve record-breaking performance, but as discussed in the previous sections, superhuman fitness for game-playing agents is not an easily spendable feature in the context of game development.

Comparisons between input representations in [49] (ALE) and [50] (Tetris) show better performance for agents evolved over game state features, as opposed to the more general and computationally heavy visual frame representation. The drawback of game state variables as input nodes is the necessity of domain knowledge. The impact of this requirement is mitigated

during the development process, as having access to the source code makes it easier to provide the necessary interface to data for agents.

6. Play for Experience

Studies in this second category are still agents playing games, although the goal differs from those described in the previous chapter. Instead of aiming to win the game, these agents aim to provide a specific experience to players. This translates to modifications in fitness function design compared to agents in the play to win category. As stated in [1], it can be argued that agents playing for experience have their roots in agents playing to win. ANNs representing agents map observations inputs to action selection outputs.

There are three main subcategories, with an evenly distributed number of studies. Work falling under the automatic playtesting subcategory focuses on evolving agents with the goal of providing a bug-free experience to players. The agent therefore needs to be competent enough to be able to complete the games or levels being tested. This leads to fitness functions measuring both in-game performance and off-game performance, which in this context equals code coverage.

Table 6.1. Play for experience studies distribution, grouped by experience type.

Experience category	N° of studies	%
Automatic playtesting	5	35.71
Difficulty control	5	35.71
NPC behavior	4	28.57

6.1 Automated playtesting

[28], [29] and [80] introduce and discuss the NEATEST framework and are limited to simple games made in Scratch. The ability of an agent to complete a given game contributes to a fitness function that rewards the exploration of different states, including the victory state, which provides a large reward. Experiments were run on 25 Scratch games with elements of randomized behavior, evolving a population of 300 with 23 hours search budget. Results show that NEATEST achieves high block coverage, does not lose coverage over re-execution compared to static test suites, and can distinguish behaviors serving as a test oracle [80]. The same authors in [29] use NEATEST integrating human gameplay traces. Results show that gameplay traces based on 1-minute-long recordings and without no-operation actions had best results for adapting input generating networks, unlike in other supervised learning approaches. Also, combining neuroevolution with gradient descent achieved significantly faster search times, while lowering the population diversity. In [28] it is stated that while NEATEST performs robustly for the Scratch games domain, more work is needed to adapt it to more complex games.

The Wuji automated playtesting framework introduced in [60] instead relies on multi-objective optimization, separating in-game performance from code coverage. Agents are randomly initialized DNNs evolved by a multi-objective EA with tournament selection, using both single-point crossover and mutation. The evolved population is then trained by the advantage actor-critic gradient-based optimizer. Experiments were run on two commercial online combat games (“A Chinese Ghost Story” and “Justice Online Treacherous Waters”) and a simple game (Block

Maze) by comparing Wuji's performance to other solutions (random, base EA, multi-objective EA, deep reinforcement learning, single fitness EA with deep reinforcement learning). The results demonstrate that Wuji achieves higher coverage compared to the alternative methods.

[70] introduces Bonobo, an adversarial behavior debugging system using AI for video games developed in the Unity game engine. The approach in this study is different, as it provides survey-based experimental results, on top of empirical data on agent behavior. It is also narrower in scope, as it aims to evaluate how exploitable an authored agent is, rather than an entire codebase. Empirical results showed that evolved agents are more consistent in their behaviors (repeated use of actions), providing different insights compared to the less predictable human playthroughs. The survey required participants to detect buggy or exploitable authored agents by watching footage of human, random, and evolved ANN agents playing against them, while also judging how useful the footage proved to be in the detection process, although no significant difference between human and ANN agents was detected in footage usefulness.

6.2 Difficulty control

In difficulty control, the idea is to evolve an agent towards a specific level of competency rather than solely maximizing its performance, potentially leveraging the model to adjust the difficulty at runtime. This introduces new challenges and complexities to fitness function design. Nevertheless, it's interesting because it opens the possibility of adapting findings on solutions in the play to win category to difficulty control.

[27] addresses the problem of game balance through neuroevolution of AI agents, using the game Ms PacMan as a testbed. Fitness evaluation of individuals is the sum of the distance between the desired and average score of the agent (over 50 games) plus the distance between the observed and desired standard deviation. Experiments aiming at maximum performance, specific average score and specific standard deviation were performed, proving this approach as a useful tool for adjusting AI skill levels. By adjusting the target standard deviation, it is possible to achieve different behaviors: a high variance led to a risky playstyle, while aiming at a lower standard deviation developed higher consistency (at the cost of search time due to the randomness of game episodes in the domain).

Similarly, in [92] a custom real-time first-person fighting game is used to test neuroevolution for dynamic difficulty adjustment. The fitness function rewards minimization of the difference between the current difficulty and the player's detected skill level. Difficulty and player skill level are both functions of damage dealt, elapsed time, health, number of hits and misses.

In [101] dynamic difficulty control is addressed with a reinforced EA based on difficulty difference (REA-DD). Two testbeds are used: the game Pong and The Ghost Story (A Chinese Ghost Story), both zero-sum games. In this study, individuals are trained using reinforcement learning and the resulting neural networks are evolved by an EA. Parent selection happens through tournament selection. Both single-point crossover and Gaussian mutation can alter individuals. The fitness evaluation is a function of the agent's win ratio against an opponent. A multi-objective variant of REA-DD is also evaluated (RMOEA-DD), with two fitness functions representing the lower and upper bound of the difficulty range. Experiments show faster convergence for RMOEA-DD compared to reinforcement learning and multi-objective evolutionary algorithms. Comparing the results, RL provides the larger contribution to faster convergence.

[71] also attempts at evolving agents to reach a desired level of skill in order to provide an adequate challenge for players. The testbed for the study is a custom-built racing game in the

Unity game engine. The fitness function for the agent depends on the agent's distance from the finish line, the elapsed time, and time spent correctly inside a lane. Agents were evolved with a GA featuring a population of 25 and elitist selection. Experiments show that it is possible to obtain diverse behaviors by adjusting weights within the fitness function definition, which is a different approach from including a measure of diversity in the fitness function. This is part of a multi-step process, where a baseline network is evolved to competently play the game and then re-trained altering the weights of the fitness function to obtain different behaviors.

In [7] introduces AI for an asymmetrical RTS game developed with the Unity game engine. A NEAT approach is evaluated along with others, using UnitySharpNEAT. The agent outputs correspond to macro-actions, not single unit actions (which are implemented through traditional AI methods). The goal of the study is to find gameplay balance in an asymmetrical setting, find the best static opponent to evolve solutions against, and find the best performing solution in direct confrontation among traditional rules-based AI, coevolved AI, and NEAT-evolved AI. Experiments show NEAT agents having the worst performance overall, although no information regarding fitness evaluation or other specifications are available, making it difficult to infer the root cause.

6.3 NPC behavior

In this last subcategory, the focus is on studies dedicated to developing agent behaviors through neuroevolution for non-playable characters (NPCs). Unlike the previous section, the goal is to provide specific experiences untied to the game's difficulty. Also, it can be observed that 3 out of 4 studies in this subcategory rely on surveys for psychological feedback.

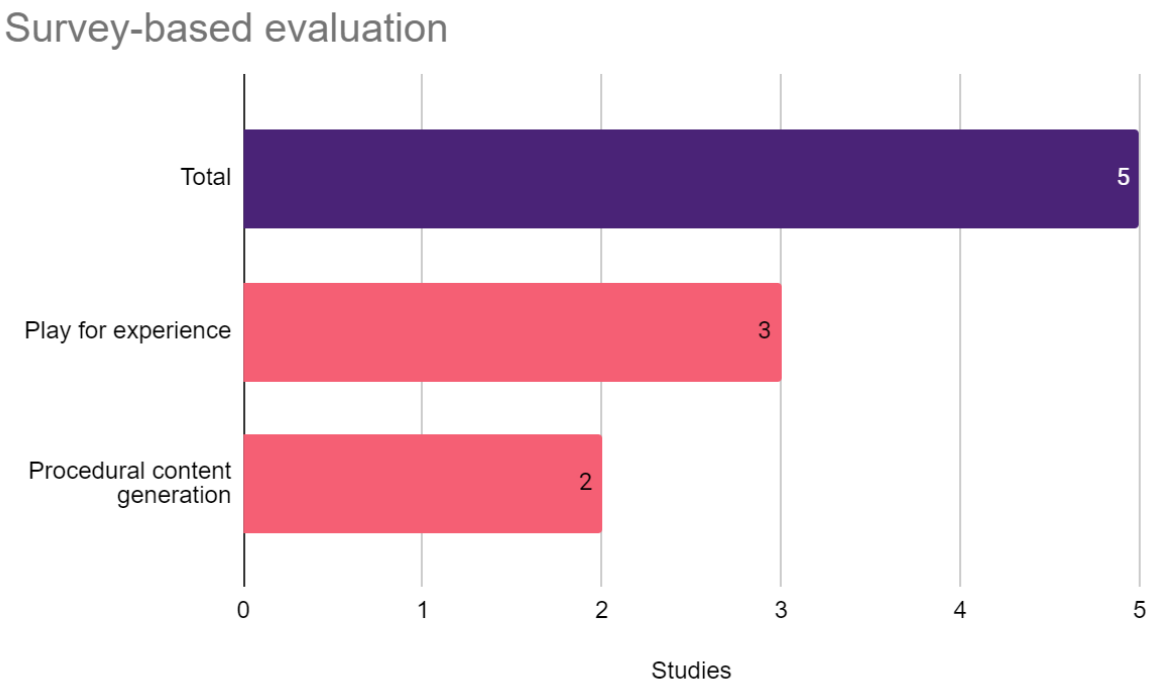

Figure 6.1: survey-based studies category distribution.

In [135] agents in a custom environment were evolved through Natural Evolution speciation NEAT (NENEAT). In this simulation, agents evolve a very simple behavior. They start at the edges of an environment and must navigate to the center, collect food pills, and get back to an edge.

[72] introduces a custom racing game developed in the Unity game engine with the more sophisticated goal of developing believable AI opponents. ANNs feature sensory inputs from the car agent, while outputs represent different actions such as accelerating, braking, and steering. Evolution algorithm is NEAT and fitness of individuals is measured through the car's racing performance, rewarding the number of completed laps and current track piece the car is in (the further the better) while punishing the number of collisions with walls. A survey on a small sample of human subjects showed that these agents could convincingly pass as human players, although no quantitative measure of this capability is available.

In [75] MM-NEAT (a NEAT variant supporting multi-objective evolution through Non-dominated Sorting Algorithm II, or NSGA-II) is used to evolve a bot playing Unreal Tournament 2004. Experiments involved human subjects playing the game with screen recordings alternating a traditional AI agent (focused on supportive teamplay) and the evolved ANN agent (focused on team performance with a fitness function that aims to maximize the team's score and damage dealt, minimizing damage taken at the same time) as teammates. Survey results indicated that players have different definitions of teamplay and enjoy different aspects of the game experience. There was not a clear preference, as some people preferred playing a more challenging game with a less competent bot, while others enjoyed the better ANN agent performance in winning the game.

[73] presents the game EvoCommander, a game where players evolve ANN-controlled behaviors for robot tanks by designing their training directly in-game. Players equip a subset of these behaviors ("brains") and switch between them in real-time to win battles. Surveys following playtests on a small sample of human subjects demonstrated that players positively engaged with the novel game mechanic.

6.4 Contributions to RQs

6.4.1 RQ1

What are the most common neuroevolution techniques in the context of developing digital and non-digital games?

Solutions in this category follow the play to win agent structure pretty closely. ANN agents translate game observations to action selection and measure their performance in the game's domain. In order to achieve the desired experiences, it can be observed that the main modifications happen at the fitness function level. Generally, fitness functions become more complex, almost always relying on guidance, as agents still need to be able to play the game at least competently to provide specific experiences. In automatic playtesting, an agent is fit if it can cover a large part of the code. Studies in the NEATEST domain measure code coverage directly. Because of the framework, this implicitly includes taking victory states into account when evaluating agents.

In the dynamic difficulty domain, performance is measured relative to a given target. In NPC behavior design agents do not differ from play to win in their implementations fitness wise. Desirable behavior stems from completing a task, and the extent to which the behavior matches

the desired output is not quantitatively measured in the fitness function but rather assessed through surveys.

Studies [101] and [60] aim at generality in their approach, being tested in multiple scenarios and leverage visual input, handled by a combination of neuroevolution and reinforcement learning, a pattern already detected in the previous chapter. Outside these exceptions, studies focus on single games and adopt the simpler game state input representation. NEAT is the most popular EA for this input representation, with a minor presence of simple GAs.

6.4.2 RQ2

What are the potential use cases for neuroevolution in the context of developing digital and non-digital games?

The sub-categories highlight three use cases for neuroevolution in games.

Automated playtesting evolves game-playing agents to help the quality assurance process and develop ANN-based test oracles. NEATEST body of work [28][29][80] has strong ties with the framework, while Wuji [60] attempts a more general, vision-based approach that relies on state exploration, therefore requires domain knowledge to model the game as a Markov Decision Process (MDP).

Dynamic difficulty adjustment evolves agents that are competent in the game and translate the in-game performance maximization problem into minimizing the distance from a desired level. Fitness is evaluated relatively to a desired level. [27] experiments with meta-information fitness goals, including control of the standard deviation of the agent over multiple runs.

Studies concentrating on attaining specific NPC behaviors do not deviate from play to win agents in their implementation. Rather, the distinction lies in how the overall solution is assessed. The evaluation of how closely the agent's behavior aligns with the desired target is not quantitatively indicated by fitness but is instead derived qualitatively from surveys involving human subjects. [73] is the only study in this macro-category building a novel game design on top of neuroevolution.

6.4.3 RQ3

What are the costs and benefits of utilizing neuroevolution in the context of developing digital and non-digital games?

Automatic playtesting in the NEATEST framework shows that neuroevolution provides benefits in terms of robustness and requires no domain knowledge to produce competent agents, although the results are not guaranteed to translate to other languages and games [80]. Wuji [60] has a more general approach, being tested on multiple games and working on visual input, although the solution combines multi-objective evolution with RL. The evolutionary aspect of the solution is used to promote diversity using the crowding distance algorithm. Similarly, in [101] a combination of neuroevolution and RL is used to address the difficulty control problem, reinforcing the concept of diversity provided by multi-objective evolution contributing to faster convergence.

The rest of the studies do not provide sufficient evidence for comparison but are of interest for feasibility in their goals and the survey-based experiments. [73] in particular is the only study leveraging neuroevolution as a design foundation in this category.

7. Procedural Content Generation

Studies in the procedural content generation macro-category are further divided by the type of game element the solution attempts to generate. The most common goal is to generate levels encoded as 2D or 3D discrete grids, although a few studies also attempt at generating NPCs and NPC behaviors, or game design elements (such as rules, systems, or their parameters). ANNs are evolved to either be generators, or agents supporting the generation process.

Table 7.1. Procedural content generation studies distribution, grouped by content type.

Content type	N° of studies	%
Levels	7	50.00
NPCs	4	28.57
Game design	2	14.29
Other	1	7.14

7.1 Levels

Procedural level generation is the most popular subcategory. Level generation ANNs map coordinate inputs (two or three-dimensional, depending on the testbed) to tile information outputs in 2D and 3D discrete grids.

In [34] coevolution of agents and environments is used to achieve complexity and diversity in procedurally generated levels. The testbed is the Griddly AI framework, a 2D grid-based game environment designed for RL. Agent ANN receives the map's grid as input, has convolutional layers for extracting features from the map, recurrent layers for memory and outputs the agent's actions. Agent evolution uses Canonical Evolutionary Strategy (CES) with a score-based fitness function. The environment ANN was represented by a Neural Cellular Automata (NCA) and evolved with an ES. Fitness in this case was the Euclidean distance between the output and a real map. During the coevolution process, environments are eliminated based on a fixed criterion (map feature density) and agents are tested on the remaining environments. Experiments compared directly evolved agents with coevolved agents on existing maps, highlighting better performance (fitness) for coevolved agents. While giving no insight on the quality or diversity of procedurally generated maps, this demonstrates the feasibility of the coevolutionary approach at PCG.

In [81] NEAT is used to train an ANN to procedurally generate levels for Maze and Super Mario Bros (SMB) without a training dataset or specific domain knowledge. This approach is named Procedural Content Generation using NEAT and novelty search (PCGNN). Experiments

compare solvability, generation time (for real-time applications), difficulty and diversity of the NEAT approach against RL and GA based approaches. Results are aligned with the baseline except generation time, which proves to be considerably faster for NEAT. Furthermore, the authors believe the domain knowledge independence of this solution to provide potential generality for this PCG method.

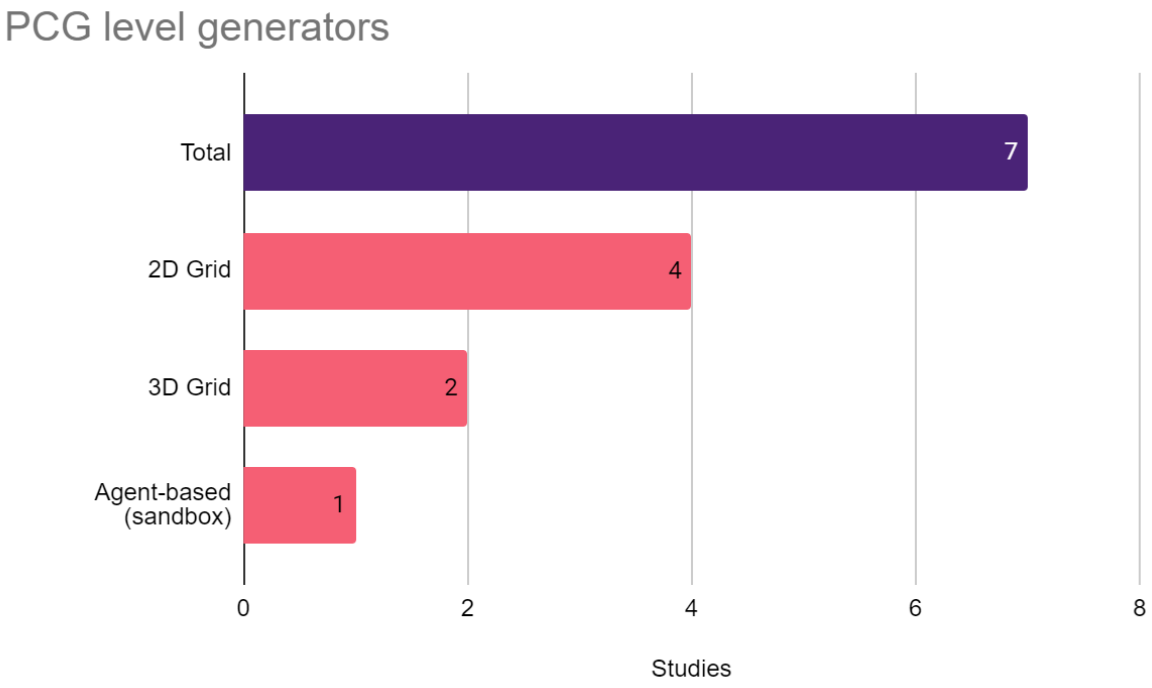

Figure 7.1: PCG level generators distribution based on generation method.

Moving from 2D environments to 3D ones, in [48] PCGNN is used within Multi-level Composition of Hierarchical and Adaptive Models via Recursion (MCHAMR), presented in the study. The solution attempts to generate levels of higher complexity compared to the 2D worlds of SMB and Maze in Minecraft. The strategy is to combine multiple simple generators each having separate and specific tasks (fitness functions). This is showcased in the study by creating towns and cities out of building blocks with their own PCGNN generator (houses, gardens, and roads).

Another Minecraft PCG approach [47] evolves CPPNs with NEAT. CPPNs encode a Boolean lattice (empty of not empty voxel) which is fed to a repair function cleaning up the structure and assigning materials to voxels. Finally, an autoencoder (represented by a CNN) compresses them in a latent vector. This final representation is used to evaluate novelty scores which are then archived. Experiments with different autoencoder training strategies yielded better results in terms of novelty compared to static scenarios.

[32] introduces Voxelbuild, a sandbox environment inspired by Minecraft where AI agents can build structures in a discrete 3D grid. The ANN is a large MLP evolved using HyperNEAT, an indirect encoding NEAT variant where a NEAT evolved CPPN assigns connection weights to different substrates. A multi-objective quality diversity algorithm (novelty search with local competition) is used to search for novelty and performance locally to a specific behavior. Experiments detect evolutionary transitions in agent behavior, key factors in open-ended evolution. Unlike other methods so far, the ANN does not directly represent a generator, but is

modeled after an agent playing a game where the action space corresponds to the atomic operations used during map creation. While conceptually similar to a generator, this approach differs in its implementation, more specifically in the input/output mapping.

The study in [31] introduces a method (CPPN2GAN) to procedurally generate large-scale levels for the 2D games Super Mario Bros and The Legend of Zelda. This approach combines a previously trained GAN and a CPPN evolved by NEAT (in its multi-objective modular variation, MM-NEAT). The CPPN generates latent vector inputs (level layouts) for the GAN to generate level segments. The method is compared with Direct2GAN, which generates a level directly from genomes subdivided in GAN inputs with a defined convention. Results from experiments with divergent search through MAP-Elites (Multi-dimensional Archive of Phenotypic Elites) showed that CPPN2GAN covers a larger portion of the design space, while Direct2GAN provides better variety within generated content, as CPPN2GAN is prone to repetition in level segments.

[59] follows up with a hybrid solution, a combination of CPPN2GAN and Direct2GAN called CPPNThenDirect2GAN. The addition of a mutation operator switching indirect encodings to direct encodings during the evolution process allows for a better localized variation.

7.2 NPCs

This section differs from evolving NPCs for specific experiences discussed in Chapter VI. The goal of these studies is still to evolve agents in the game world, although the behaviors are not necessarily specific or tied to a desired experience.

In [36] the study presents Automatic Generation of Interrelated Organisms (AGIO), an artificial life generation framework. An organism's representation in AGIO is a combination of a morphology (tags defining sensors and possible actions), parameters (multiple mandatory and optional numerical values), and an ANN. The system is high level and does not depend on the representations of its elements. Experiments in a predator-prey setup showed that under similar circumstances AGIO organisms behave similarly to human players fitness wise. Real-time computational performance on consumer hardware was achieved.

The study [37] attempts to procedurally generate a communication language for AI agents. Speakers and listeners are both represented by RNNs, with two main architectures being tested: Gated Recurrent Units (GRUs) and partially randomized RNNs. Both types of agents use a FNN to preprocess inputs for the RNN. Experiments run on a dataset on evolved and unevolved versions of both the ANN architectures showed better performance for their evolved counterparts, with GRUs performing better than partially randomized RNNs. Still, neuroevolution performed worse than deep reinforcement learning in this benchmark.

7.3 Game design

A couple studies apply neuroevolution for designing entire games or tuning parameters of design elements. In this context, game design elements refer to rules, mechanics and systems pertaining to a game (what combination and number of enemies are part of an encounter in a combat focused game? What kind of movement actions are available to the player? Etc.).

This is the case in [32], where neuroevolution helps generate enemy waves in a tower defense game developed with the Unity game engine. An ANN evolved with NEAT functions as a wave manager, capable of spawning different assortments of enemy units for the player to face. A/B testing on human subjects was used for evaluation of player engagement and content generation

quality. The study's findings revealed no significant variations in engagement, and the survey data comparing procedurally generated content to human-designed content did not show a statistically significant difference. This suggests that procedurally generated content can serve as an alternative without requiring hands-on development.

[74] explores the procedural generation of games in the endless runner genre. In these games, players navigate a continuously side-scrolling environment with the objective of progressing as far as possible while avoiding a variety of obstacles. To achieve this, three populations coevolve: level generators, player controllers, and game descriptions. Only player controllers are represented by ANNs. Game descriptions are permutations and combinations of pre-designed features, including speed altering effects, or special blocks with on-collision effects. Evolved agents' fitness competes with level generators and game descriptions to generate interesting games. Experiments show that this method is capable of generating diverse games with a variety of mechanics, although game descriptions tend to be noisy, which in this case means that the games sometimes feature mechanics with no real impact on the experience.

In [30], the authors developed the Infinite Art Gallery in the Unity game engine. It is an open-ended first-person exploration game where players navigate through multiple rooms featuring procedurally generated artwork. The artwork also serves as a traveling system, influencing the content generated in each room based on the content used for travel. A survey on human subjects was conducted. Results showed that the game experience provided enjoyment, but the influence of evolution on the 3D sculptures felt lacking compared to 2D artwork.

7.4 Contributions to RQs

7.4.1 RQ1

What are the most common neuroevolution techniques in the context of developing digital and non-digital games?

Two main solution architectures can be observed in procedural content generation: pure generator networks and agent networks. Agent networks follow the structure already encountered so far in both Chapters V and VI. They can be found representing NPCs, where the evolution of diverse individuals capable of enacting game-related behaviors corresponds to the generation process itself [34][36][37], or they can play a part in a coevolutionary setup [74][34].

Generator networks are most often used for level generation. More specifically, the level representation is a discrete 2D or 3D grid, depending on the game domain. These networks often leverage CPPNs as base networks, leveraging their structural regularities for generating patterns. NEAT and its extensions are the most popular choice, and they are often used in combination with other methods (novelty search in PCGNN [81][48] and GANs in CPPN2GAN [59][31]). The input to output mapping in this kind of network produces content features for a given tile or block based on its coordinates. The novelty meta-information is often used for level generation, where fitness is more difficult to measure quantitatively and typically includes the generated level's feasibility.

7.4.2 RQ2

What are the potential use cases for neuroevolution in the context of developing digital and non-digital games?

The single most common use case for procedural generation is level generation. Both 2D and 3D level generation through neuroevolution is achieved, although the representations are limited to discrete grids. Hierarchical level generation in [48] shows that abstraction and separation of context can be used to evolve more complex designs.

Procedural generation of NPCs with neuroevolution is not supported by as much research. [36] proposes a game-agnostic neuroevolution-based system for developing entities capable of interacting with each other in a simulation environment. The experimental implementation focuses on a predator-prey setup. The attempt at generating procedural languages in [37] performs worse than its RL counterpart.

ANNs can be evolved to design game elements like enemy waves in a tower defense game and provide an equivalent experience to hand-crafted ones, according to the survey in [32]. Agent ANNs also can help procedurally generate game prototypes like shown in [74], although generation from scratch would require finding a suitable encoding of game systems and rules, as well as a measure of fitness.

7.4.3 RQ3

What are the costs and benefits of utilizing neuroevolution in the context of developing digital and non-digital games?

Neuroevolution shows some versatility in procedural content generation, addressing multiple types of elements.

PCGNN [81][48] shows better level generation times than GA and RL baselines and is able to scale to arbitrary level sizes without retraining. The method lacks domain knowledge, so not all levels are usable. The usage of novelty search and MAP-Elites contributes to open-endedness by promoting exploration and diversity preservation.

8. Player Modeling

The last and smallest category is dedicated to solutions attempting to use neuroevolution to model or imitate existing behaviors, and in preference learning.

In [131] statistical penalties are given to a computer player to punish mechanical playing as opposed to human-like playing in the Mario AI Benchmark. Threshold values are extracted from human player data and are used in the fitness evaluation of a $(1 + \lambda)$ evolution strategy. Evaluation of human-likeness is performed by a small sample of human subjects. Experiments show an improvement thanks to statistical penalties, but the agent didn't reach human-like performance.

[102] features evolutionary adversarial generation on multiple videogame testbeds (Pong, LunarLander, Shinobi III). The ANN architecture used for generators and discriminators (trained on a gameplay dataset) is a dynamic convolutional network connected to a dynamic recurrent network. Fitness is evaluated for score maximization and authenticity confidence (for GANs imitating previously RL trained agents). Experiments comparing static and dynamic setups and both fitness evaluation showed that dynamic networks outperformed static networks and score optimization outperformed behavior imitation.

Another approach to this method can be found in [79]. In the study two populations of individuals are coevolved to imitate high performance RL pre-trained static deep RNN agents activated by ReLU. In this GAN setup, generators are randomly matched with discriminators and their fitness is evaluated based on how incorrect the paired discriminator assessment is and discriminators have their fitness evaluated on how correct their assessment is. Evolution is based on a simple GA with no crossover or speciation methods. Experiments proved successful on various control tasks, the game LunarLander being one of them. Imitated behaviors match their targets in final score and score trajectory.

In [38] RankNEAT is compared to Stochastic Gradient Descent (SGD) in the context of preference learning, specifically predicting annotated player arousal from game footage of three different games in the AGAIN dataset. A vision transformer is used for dimensional reduction of gameplay footage frames.

8.1 Contributions to RQs

8.1.1 RQ1

What are the most common neuroevolution techniques in the context of developing digital and non-digital games?

NEAT-based algorithms are still the most common choice for this small subset of studies, with a single GA exception. [79] and [102] combine GANs with evolutionary algorithms to imitate pre-trained RL agents. Fitness for agents is often characterized by a measure of how close the agent's behavior is to a pre-trained RL agent or data extracted from an existing dataset.

8.1.2 RQ2

What are the potential use cases for neuroevolution in the context of developing digital and non-digital games?

Behavior imitation through neuroevolution in these studies does not achieve successful results in experiments run in studies under review. Opponent modeling in [54] and preference learning in [38] show better results in their goals. It can be observed that these solutions leverage neuroevolution in combination with other methods to reach their goals (pattern recognition tree and visual transformer for dimensionality reduction respectively).

8.1.3 RQ3

What are the costs and benefits of utilizing neuroevolution in the context of developing digital and non-digital games?

RankNEAT in [38] can find increasingly accurate models compared to the SGD baseline by being less prone to overfitting. GAN-based coevolutionary solutions require pre-trained RL agents and their performance is bound to them, making it less appealing for game development. Similarly, [131] requires an existing dataset, which might not always be available during the development process.

9. Conclusions

The analysis of the 118 selected pieces of work under review provided valuable insights into the current research around neuroevolution in games. After addressing the research questions and discussing open challenges, limitations of this work will be discussed.

9.1 RQ1

What are the most common neuroevolution techniques in the context of developing digital and non-digital games?

In the vast majority of studies, the ANNs represent agents playing a game. Agents represented by ANNs receive observations of the game world as input. These observations can be subdivided in three main categories: visual frames, features extracted from game states, and RAM states. Agents relying on RAM states are few and are tied to the emulator-based frameworks that offer such representations, such as the ALE or NES/SNES emulators. This information requires domain knowledge to annotate these states, creating a representation that closely aligns with game state features. Simultaneously, it maintains a degree of generality by adopting a similar structure across different games on the same platform due to hardware restrictions, akin to the standardized format found in visual frames. Visual frames tend to be used in solutions attempting at some degree of generality by dealing with multiple games. Another common trait in these solutions is their high dimensionality, which leads to common occurrences of DNNs as starting networks and various methods of dimensionality reduction for the input, such as autoencoders or self-attention. While the ability to play different games over visual input can hardly be the most efficient method to employ in game development (but has value in researching general intelligence), the algorithmic findings still provide valuable insights.

LSTM cells find success in imperfect information games, as well as racing games. These cells are also core to the architecture of OpenAI Five, a large-scale DRL system that defeated the DotA 2 world champions [140]. DotA 2 is a high complexity team-based real-time strategy game with imperfect information (due to fog of war), well beyond that of the games in studies under review. [55] proved that memory modules can be evolved end-to-end together with other modules (vision and decision-making in the study).

In a few studies, the agent is used as a state evaluator rather than for direct action selection, which is the absolute most common output type for agent ANNs (represented either as controller input or in-game action).

Fitness for agents involves in-game performance metrics in most cases. This depends on the specific game domain, but there are some patterns shared between genres. Games with combat elements consider an agent's health and possibly an opponent's health, games with navigation goals reward travel distance towards the goal, score-based arcade game reward scoring actions, gambling games measure the return of interest.

Very often additional in-game metrics are added in order to provide guidance to the agent, promoting or discouraging specific actions (or inaction, as it is often the case) that do not directly relate to the agent's goal. This narrows the search space for fit solutions, saving iterations. Another approach is multi-objective optimization, that separates different

(potentially conflicting) fitness functions and ranks individuals accordingly, aiming at the Pareto front. NSGA-II is the most popular sorting algorithm of choice.

In automated game testing, behavior imitation and preference learning agents are evaluated in their performance outside the game (e.g. code coverage, imitation confidence, etc.).

Another useful technique in fitness function design observed in the work pool is the usage of the novelty meta-information. Novelty measures the difference between individuals to favor exploration in the solution space to escape local optima and overfitting. This is typically achieved through a distance function. Novelty search with local competition and MAP-Elites are common implementations. In competitive coevolution, fitness is computed in a relative manner between individuals.

A branch of studies on the Atari 2600 benchmark extend the ES described in the seminal work [45], with CMA-ES being a very common choice. The most common family of evolutionary algorithms is based on NEAT. Its modular nature and numerous extensions make it a versatile and efficient solution to a variety of problems. Neuroevolution can be also found in several studies featuring modular or hybrid solutions, mixed with various other methods.

Outside of game-playing agents, evolved ANNs can also be used as procedural content generators. The most common occurrence is level generation of discrete grid-based levels, where the ANN inputs are coordinates and the output encodes content features (e.g. tile information). A single instance of pure function approximation (player arousal prediction) can be found in [38], for preference learning in player modeling.

9.2 RQ2

What are the potential use cases for neuroevolution in the context of developing digital and non-digital games?

Playing to win might be considered the main goal for AI in games since its birth. As described in [1], among the first breakthroughs in AI reaching superhuman performance dates back to 1994 with Checkers and to 1997 with the famous chess game between IBM's Deep Blue and Garri Kasparov. It can be argued that in the game development process, superhuman performance for AI agents playing against human players is of limited utility. Even in studies like [7], where exploitative agents are used to test game balance, the convergence to imbalance might happen at a level that is already unreachable by human players. Results in the play for experience category, especially those in the dynamic difficulty adjustment subcategory, show that the solutions are structurally the same as those in the play to win, except for modifications on the fitness function design.

Game-playing agents in studies under review have evolved in relatively simple game domains. Flappy Bird is among the most basic games in the pool, with its single input, single goal and failure condition. ATARI 2600 games have solution spaces of varying deceptiveness, but games are often simple and limited by the hardware capabilities. Racing games, especially the popular 2D top-down CarRacing, also involve a simple goal and a limited action space. SMB is slightly more complex in its action space and environment, while the goal is still simple (travel to the right). The Mega Man II inspired EvoMan framework limits the action platformer title to its boss fight sequences, making it closer to a fighting game conceptually. The VizDoom framework is also used as a testbed in specifically designed levels that are simpler in nature than the original Doom game it is based on. RTS games are also represented by solutions (based on StarCraft: Brood War and StarCraft II) that employ ANN agents for the control of combat

units. This is only a limited and specific subset of the complex goals and action space of a RTS game.

Automated playtesting for quality assurance and development of ANN-based test oracles in studies rely on the NEATEST framework, which narrows down its use-cases. A more generalized vision-based approach is also introduced in [60], which requires domain knowledge. If this need is anticipated and addressed early in the game development process, it does not pose a significant issue.

Procedural content generating ANN evolved through neuroevolution has use cases in 2D and 3D discrete grids, and [48] introduces hierarchical level generation through a methodology that has potential applications in other neuroevolutionary solutions. Procedural generation of NPC behaviors is also a possibility, although the experiments in studies are based on extremely simple behaviors in custom environments.

Another use case for neuroevolution in games is player modeling. As for automated playtesting, planning the game development process for this use case allows to integrate the necessary domain knowledge for behavior imitation and preference learning, as these methods require a dataset.

[73] introduces EvoCommander, one of two studies (the other being the sandbox experience in [30]) under review where neuroevolution of agents is a central element of game design, which raises the question of what might be hindering this kind of experimentation.

9.3 RQ3

What are the costs and benefits of utilizing neuroevolution in the context of developing digital and non-digital games?

The variety of technical implementations and different game domains make it different to position the performance of neuroevolutionary techniques in games or comparing the influence of single elements when solutions are composed of several moving parts. Knowledge on state-of-the-art solutions for each of the subcategories would also be necessary if available, but it is outside of the scope of this review.

In the Flappy Bird domain, it is easy to evolve agents able to play the game indefinitely. In the ALE framework, neuroevolution agents do not achieve record breaking performance, although they can best human performance in a few instances. As discussed in the previous sections, it can be assumed that in the game development process neither superhuman performance nor general vision-based game playing are desirable features for AI agents. Experiments in [49] compare various evolutionary algorithms and input representations in ALE and show that NEAT is the best performing EA and game state features are preferable due to their low dimensionality. This result can also be observed in many studies that focus on single games: low dimensionality game state features are the preferred input representation when the study is not bound to visual input by design for generality.

[27] demonstrates that achieving robustness in evolved solutions, with a focus on maintaining consistent performance, can be achieved by using the standard deviation over episodes as a fitness measure. However, computational costs can escalate depending on the stochastic nature of the environment.

9.4 Open challenges

The studies under review introduce a variety of neuroevolutionary methods for several different tasks. Often, these tasks structurally involve a game-playing agent. One of the contributions of this work is addressing open challenges in neuroevolution in the context of developing digital and non-digital games.

9.4.1 Address higher complexity game domains

As observed in Chapter V, the game domains in which neuroevolution has been applied are simple in their goals and action space, or reduced versions or subsystems of more complex games (such as unit combat in RTS games). One of the main open challenges for neuroevolution is tackling more complex game domains or game-related problems. This is a goal for future work shared by many studies under review [46][85][59][7][36][78][34][32][75][49][42][82][127][55][74]. This might possibly be achieved by sheer computational power, but there are other strategies to increase the efficiency of neuroevolutionary methods and the complexity of the evolved networks. An approach that proved to be successful and possibly warrants further exploration is the hybridization of neuroevolution and other algorithms, as mentioned in several studies [62][119][44][45][36][87]. This is also related to modular solutions with separation of contexts, such as hierarchical solutions or vision-based agents relying on dimensionality reduction. Further investigation of methods similar to [51][48][9] can shed more light on how problems can be decomposed and more complex behaviors can be evolved by decoupling high level decision making from single task learning (similarly to how vision based networks rely on modules or components extracting features from high dimensional and noisy raw input), obtaining at the same time more readability for solutions as mentioned in [9]. An unsuccessful attempt is made in [7], but more work and experiments might be necessary to further evaluate the feasibility of such an approach. LSTM cells proved to be beneficial in imperfect information games [52][51][23][53] as well as racing games [55][56], and [55] proved that they can also be evolved end-to-end in modular solutions. This architecture can be tested on more games, since action sequences are commonly found in a large number of game domains.

9.4.2 Examine in depth the impact of fitness function design

This leads to another important topic that emerged from the review, which is the importance of further exploring the impact of fitness function design, also mentioned as a direction for future work in various occasions [24][81][23][8][76][118][28][113][77]. Several studies under review leverage guidance and multi-objective fitness function design to converge faster to competent or desirable behaviors. Comparing different approaches might give valuable insight on the impact of these kinds of decisions. Further exploring the usage of meta-information in the form of novelty [77][57][106], through novelty search and quality diversity algorithms, is also a powerful tool that can help with overfitting while also being an interesting alternative approach for experimentation with procedural content generation and emergent behaviors.

Hints of patterns in fitness function design and its relationship with game genres can already be observed within this review, but expanding this knowledge might also help better understand the more deceptive game domains and improve our comprehension of game design elements. Coevolution of agents and environments [34][74][64] can also be the testbed for inquiring the relationship between level design and agent competency (for example, comparing different definitions of leniency and how they influence an agent's evolution).

9.4.3 Measure hyperparameters tuning overhead

While the solutions under review allow evolution to happen hands-free, the overhead of the hyperparameter tuning process is rarely discussed. Hyperparameters are often listed, but rarely compared (as they can vary for different methods). A few studies share this concern [76][63][113]. Furthermore, the trial-and-error process that leads to the definition of the final configuration of hyperparameters used in experiments is rarely discussed, but its importance might be not negligible in a game's production process, especially considering the iterative nature of game development.

9.4.4 Explore novel game designs

Neuroevolution-based game mechanics are definitely an under-researched niche. Several NEAT implementations are available for the widely adopted Unity Game Engine, some of which have been used in studies analyzed in this review [70][7][71][72][73][32][103], which possibly rules out the unavailability of neuroevolution methods and implementations among game development tools. On top of leveraging the versatile procedural content generation neuroevolution can achieve, this technology has exciting possibilities in evolving emergent behaviors, especially through novelty and open-endedness. This approach could possibly lend itself well to sandbox and simulation games, as the potential unpredictability of this kind of game-playing agents can favor these genres more than traditional game designs.

9.5 Brief comparison with previous state of the art

Although the work by Risi and Togelius [2] does not employ the methods of a SLR, it can still be of interest to compare the findings of this study to the 2017 survey. The selected study in [2] also features game-playing ANN agents evaluated over their performance in-game most prominently. The usage of guidance in fitness function design is uncommon in the sample of studies, and while novelty had already been introduced by Lehman and Stanley in 2011 [139] it was yet to be adopted by neuroevolution-based solutions in the games domain. More use cases for neuroevolution in games have emerged, such as those in the play for experience category, and more sophisticated PCG methods. On the other hand, a few more novel and experimental game ideas were cited between NERO [132], GAR [133], and Petalz, [134] compared to [30] and [73].

9.6 Limitations in study design

While the application of evolutionary algorithms to neural networks, or neuroevolution, is a subset of machine learning and its application to games narrows it further down, the breadth of the topic is still quite relevant. In part due to games taxonomy still being a research topic in a relatively young field (that of game studies), there might be overlapping areas. Some distinctions might be subtle, especially for evolving agents which can, depending on their fitness measure and role within a composite solution, blur between playing for a specific experience, procedurally generate diverse behaviors or imitate existing behaviors.

10. References

[1] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. 2019. Accessed: Oct. 28, 2023. [Online]. Available: https://link.springer.com/book/10.1007/978-3-319-63519-4

[2] S. Risi and J. Togelius, "Neuroevolution in Games: State of the Art and Open Challenges," *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 1, pp. 25–41, Mar. 2017, doi: 10.1109/TCIAIG.2015.2494596.

[3] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep Learning for Video Game Playing," *IEEE Trans. Games*, vol. 12, no. 1, pp. 1–20, Mar. 2020, doi: 10.1109/TG.2019.2896986.

[4] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. in Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. doi: 10.1007/978-3-662-44874-8.

[5] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002, doi: 10.1162/106365602320169811.

[6] A. Abraham and L. Jain, "Evolutionary Multiobjective Optimization," in *Evolutionary Multiobjective Optimization*, Springer, London, 2005.

[7] E. Suchardova, "Artificial Intelligence for a Castle Conquest Simulation Game," 2021.

[8] D. Dyankov, S. D. Riccio, G. Di Fatta, and G. Nicosia, "Multi-task Learning by Pareto Optimality," in *Machine Learning, Optimization, and Data Science*, vol. 11943, G. Nicosia, P. Pardalos, R. Umeton, G. Giuffrida, and V. Sciacca, Eds., in Lecture Notes in Computer Science, vol. 11943. , Cham: Springer International Publishing, 2019, pp. 605–618. doi: 10.1007/978-3-030-37599-7_50.

[9] K. O. Ellefsen and J. Torresen, "Self-Adapting Goals Allow Transfer of Predictive Models to New Tasks." arXiv, May 15, 2019. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/1904.02435

[10] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *jair*, vol. 47, pp. 253–279, Jun. 2013, doi: 10.1613/jair.3912.

[11] S. Karakovskiy and J. Togelius, "The Mario AI Benchmark and Competitions," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 55–67, Mar. 2012, doi: 10.1109/TCIAIG.2012.2188528.

[12] F. O. de Franca, D. Fantinato, K. Miras, A. E. Eiben, and P. A. Vargas, “EvoMan: Game-playing Competition.” arXiv, Jan. 04, 2020. Accessed: Mar. 24, 2024. [Online]. Available: http://arxiv.org/abs/1912.10445

[13] J. Laserna, A. Otero, and E. D. L. Torre, “A Multi-FPGA Scalable Framework for Deep Reinforcement Learning Through Neuroevolution,” in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, vol. 13569, L. Gan, Y. Wang, W. Xue, and T. Chau, Eds., in Lecture Notes in Computer Science, vol. 13569. , Cham: Springer Nature Switzerland, 2022, pp. 47–61. doi: 10.1007/978-3-031-19983-7_4.

[14] U. Menon and A. Menon, “An Efficient Application of Neuroevolution for Competitive Multiagent Learning,” *TMLAI*, vol. 9, no. 3, pp. 1–13, May 2021, doi: 10.14738/tmlai.93.10149.

[15] F. Q. Jr. Hickam, “Automated Machine Learning Solutions to Gaming Applications,” 2017.

[16] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning.” arXiv, Sep. 20, 2016. Accessed: Nov. 28, 2023. [Online]. Available: http://arxiv.org/abs/1605.02097

[17] S. Alvernaz and J. Togelius, “Autoencoder-augmented neuroevolution for visual doom playing,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, New York, NY, USA: IEEE, Aug. 2017, pp. 1–8. doi: 10.1109/CIG.2017.8080408.

[18] F. Bloom, “Competitive Coevolution for micromanagement in StarCraft Brood War,” 2017.

[19] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, “Fighting game artificial intelligence competition platform,” in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, Tokyo, Japan: IEEE, Oct. 2013, pp. 320–323. doi: 10.1109/GCCE.2013.6664844.

[20] S. Künzel and S. Meyer-Nieberg, “Coping with opponents: multi-objective evolutionary neural networks for fighting games,” *Neural Comput & Applic*, vol. 32, no. 17, pp. 13885–13916, Sep. 2020, doi: 10.1007/s00521-020-04794-x.

[21] F. Waris and R. Reynolds, “Neuro-evolution using game-driven cultural algorithms,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Cancún Mexico: ACM, Jul. 2020, pp. 1857–1865. doi: 10.1145/3377929.3398093.

[22] S. Künzel and S. Meyer-Nieberg, “Evolving Artificial Neural Networks for Multi-objective Tasks,” in *Applications of Evolutionary Computation*, vol. 10784, K. Sim and P. Kaufmann, Eds., in Lecture Notes in Computer Science, vol. 10784. , Cham: Springer International Publishing, 2021, pp. 671–686. doi: 10.1007/978-3-319-77538-8_45.

[23] J. Xu, J. Chen, and S. Chen, “Efficient Opponent Exploitation in No-Limit Texas Hold’em Poker: A Neuroevolutionary Method Combined with Reinforcement Learning,” *Electronics*, vol. 10, no. 17, p. 2087, Aug. 2021, doi: 10.3390/electronics10172087.

[24] R. Gallotta and R. Capobianco, “Tafl-ES: Exploring Evolution Strategies for Asymmetrical Board Games,” in *AIxIA 2021 – Advances in Artificial Intelligence*, vol. 13196, S. Bandini, F. Gasparini, V. Mascardi, M. Palmonari, and G. Vizzari, Eds., in Lecture Notes in Computer Science, vol. 13196. , Cham: Springer International Publishing, 2022, pp. 46–58. doi: 10.1007/978-3-031-08421-8_4.

[25] B. A. Qader, K. H. Jihad, and M. R. Baker, “Evolving and training of Neural Network to Play DAMA Board Game Using NEAT Algorithm,” *IJCAI*, vol. 46, no. 5, Mar. 2022, doi: 10.31449/inf.v46i5.3897.

[26] JNTUA, M. S. Babu, and E. K. Reddy, “Investigations on Improvised Neural Network Chess Engine for Augmenting Topologies,” *IJMTT*, vol. 48, no. 3, pp. 214–217, Aug. 2017, doi: 10.14445/22315373/IJMTT-V48P530.

[27] M. Morosan and R. Poli, “Evolving a Designer-Balanced Neural Network for Ms PacMan,” Sep. 2017.

[28] P. Feldmeier, “Fully Automated Game Testing via Neuroevolution,” in *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Dublin, Ireland: IEEE, Apr. 2023, pp. 486–488. doi: 10.1109/ICST57152.2023.00058.

[29] P. Feldmeier and G. Fraser, “Learning by Viewing: Generating Test Inputs for Games by Integrating Human Gameplay Traces in Neuroevolution.” Apr. 13, 2023. doi: 10.1145/3583131.3590448.

[30] B. Hollingsworth and J. Schrum, “Infinite Art Gallery: A Game World of Interactively Evolved Artwork,” in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand: IEEE, Jun. 2019, pp. 474–481. doi: 10.1109/CEC.2019.8790370.

[31] J. Schrum, V. Volz, and S. Risi, “CPPN2GAN: combining compositional pattern producing networks and GANs for large-scale pattern generation,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, Cancún Mexico: ACM, Jun. 2020, pp. 139–147. doi: 10.1145/3377930.3389822.

[32] D. Hind and C. Harvey, “A NEAT Approach to Wave Generation in Tower Defense Games,” in *2022 International Conference on Interactive Media, Smart Systems and Emerging Technologies (IMET)*, Limassol, Cyprus: IEEE, Oct. 2022, pp. 1–8. doi: 10.1109/IMET54801.2022.9929595.

[33] J. K. Pugh, L. B. Soros, R. Frota, K. Negy, and K. O. Stanley, “Major evolutionary transitions in the Voxelbuild virtual sandbox game,” in *Proceedings of the 14th European Conference on Artificial Life ECAL 2017*, Lyon, France: MIT Press, Sep. 2017, pp. 553–560. doi: 10.7551/ecal_a_088.

[34] E. Chigot and D. G. Wilson, “Coevolution of neural networks for agents and environments,” in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Boston Massachusetts: ACM, Jul. 2022, pp. 2306–2309. doi: 10.1145/3520304.3534047.

[35] V. Bulitko, M. Walters, M. Cselinacz, and M. R. Brown, “Evolving NPC Behaviours in A-life with Player Proxies,” 2018.

[36] S. Pacheco, N. Ottonello, and S. Nesmachnow, “Automatic Generation of Interrelated Organisms on Virtual Environments,” in *Advances in Artificial Intelligence*, vol. 12882, E. Alba, G. Luque, F. Chicano, C. Cotta, D. Camacho, M. Ojeda-Aciego, S. Montes, A. Troncoso, J. Riquelme, and R. Gil-Merino, Eds., in Lecture Notes in Computer Science, vol. 12882. , Cham: Springer International Publishing, 2021, pp. 119–128. doi: 10.1007/978-3-030-85713-4_12.

[37] J. Sirota, V. Bulitko, M. R. G. Brown, and S. Poo Hernandez, “Towards Procedurally Generated Languages for Non-playable Characters in Video Games,” in *2019 IEEE Conference on Games (CoG)*, London, United Kingdom: IEEE, Aug. 2019, pp. 1–4. doi: 10.1109/CIG.2019.8848093.

[38] K. Pinitas, K. Makantasis, A. Liapis, and G. N. Yannakakis, “RankNEAT: outperforming stochastic gradient search in preference learning tasks,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Boston Massachusetts: ACM, Jul. 2022, pp. 1084–1092. doi: 10.1145/3512290.3528744.

[39] Y. Iwasaki and K. Hasebe, “A Framework for Generating Playstyles of Game AI with Clustering of Play Logs:,” in *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*, Online Streaming, --- Select a Country ---: SCITEPRESS - Science and Technology Publications, 2022, pp. 605–612. doi: 10.5220/0010869500003116.

[40] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[41] A. Anand, E. Racah, and S. Ozair, “Unsupervised State Representation Learning in Atari,” 2019.

[42] A. Tupper and K. Neshatian, “Evolving neural network agents to play atari games with compact state representations,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Cancún Mexico: ACM, Jul. 2020, pp. 99–100. doi: 10.1145/3377929.3390072.

[43] S. Rodríguez, F. Parodi, and S. Nesmachnow, “Parallel evolutionary approaches for game playing and verification using Intel Xeon Phi,” *Journal of Parallel and Distributed Computing*, vol. 133, pp. 258–271, Nov. 2019, doi: 10.1016/j.jpdc.2018.07.010.

[44] O. Lillerovde, “AI and Game complexity Game benchmarking by using reinforcement learning.” 2020.

[45] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning.” arXiv, Sep. 07, 2017. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/1703.03864

[46] A. Tupper and K. Neshatian, “Evolutionary Reinforcement Learning for Vision-Based General Video Game Playing,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Cancún Mexico: ACM, Jul. 2020, pp. 99–100. doi: 10.1145/3377929.3390072.

[47] M. Barthet, A. Liapis, and G. N. Yannakakis, “Open-Ended Evolution for Minecraft Building Generation,” *IEEE Trans. Games*, pp. 1–10, 2022, doi: 10.1109/TG.2022.3189426.

[48] M. Beukman *et al.*, “Hierarchically Composing Level Generators for the Creation of Complex Structures.” arXiv, Jul. 19, 2023. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/2302.01561

[49] J. N. Kielmann, “Neuro-evolution as an Alternative to Reinforcement Learning for Playing Atari Games,” 2020.

[50] L. E. Gillespie, G. R. Gonzalez, and J. Schrum, “Comparing direct and indirect encodings using both raw and hand-designed features in tetris,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Berlin Germany: ACM, Jul. 2017, pp. 179–186. doi: 10.1145/3071178.3071195.

[51] C. Tutum, S. AbdulQuddos, and R. Miikkulainen, “Generalization of Agent Behavior through Explicit Representation of Context,” in *2021 IEEE Conference on Games (CoG)*, Copenhagen, Denmark: IEEE, Aug. 2021, pp. 1–7. doi: 10.1109/CoG52621.2021.9619141.

[52] J. Luo, W. Zhang, W. Yuan, and J. Chen, “DCLN: Diverse Curriculum Learning with Neuroevolution in Six-Player Poker Tournament,” in *2021 China Automation Congress (CAC)*, Beijing, China: IEEE, Oct. 2021, pp. 2448–2452. doi: 10.1109/CAC53003.2021.9727883.

[53] W. Yuan, S. Chen, P. Li, and J. Chen, “Ensemble strategy learning for imperfect information games,” *Neurocomputing*, vol. 546, p. 126241, Aug. 2023, doi: 10.1016/j.neucom.2023.126241.

[54] X. Li and R. Miikkulainen, “Opponent modeling and exploitation in poker using evolved recurrent neural networks,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Kyoto Japan: ACM, Jul. 2018, pp. 189–196. doi: 10.1145/3205455.3205589.

[55] S. Risi and K. O. Stanley, “Deep neuroevolution of recurrent and discrete world models,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Prague Czech Republic: ACM, Jul. 2019, pp. 456–462. doi: 10.1145/3321707.3321817.

[56] Y. Tang, D. Nguyen, and D. Ha, “Neuroevolution of self-interpretable agents,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, Cancún Mexico: ACM, Jun. 2020, pp. 414–424. doi: 10.1145/3377930.3389847.

[57] S. M. Mercado, “NEAT implementation for adapting neural networks applied to ATARI Asteroids,” Jan. 2023.

[58] J. Schrum, “Evolving indirectly encoded convolutional neural networks to play tetris with low-level features,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Kyoto Japan: ACM, Jul. 2018, pp. 205–212. doi: 10.1145/3205455.3205459.

[59] J. Schrum, B. Capps, K. Steckel, V. Volz, and S. Risi, “Hybrid Encoding For Generating Large Scale Game Level Patterns With Local Variations,” *IEEE Trans. Games*, vol. 15, no. 1, pp. 46–55, Mar. 2023, doi: 10.1109/TG.2022.3170730.

[60] Y. Zheng *et al.*, “Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA: IEEE, Nov. 2019, pp. 772–784. doi: 10.1109/ASE.2019.00077.

[61] D. Klijn and A. E. Eiben, “A coevolutionary approach to deep multi-agent reinforcement learning.” arXiv, Apr. 13, 2021. Accessed: Oct. 31, 2023. [Online]. Available: http://arxiv.org/abs/2104.05610

[62] C. Carvelli, D. Grbic, and S. Risi, “Evolving hypernetworks for game-playing agents,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Cancún Mexico: ACM, Jul. 2020, pp. 71–72. doi: 10.1145/3377929.3389961.

[63] M. G. Cordeiro, P. B. S. Serafim, Y. L. B. Nogueira, C. A. Vidal, and J. B. Cavalcante Neto, “A Minimal Training Strategy to Play Flappy Bird Indefinitely with NEAT,” in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Rio de Janeiro, Brazil: IEEE, Oct. 2019, pp. 21–28. doi: 10.1109/SBGames.2019.00014.

[64] J. Flimmel, “Coevolution of AI and level generation for Super Mario game,” 2020.

[65] W. Hardwick-Smith, Y. Peng, G. Chen, Y. Mei, and M. Zhang, “Evolving Transferable Artificial Neural Networks for Gameplay Tasks via NEAT with Phased Searching,” in *AI 2017: Advances in Artificial Intelligence*, vol. 10400, W. Peng, D. Alahakoon, and X. Li, Eds., in Lecture Notes in Computer Science, vol. 10400. , Cham: Springer International Publishing, 2017, pp. 39–51. doi: 10.1007/978-3-319-63004-5_4.

[66] M. McBrien, A. Melehan, and M. Munns, “LEARNING PONG.” Apr. 21, 2021.

[67] G.-C. Cojocaru, S.-A. Dinu, and E. Croitoru, “Increasing the Upper Bound for the EvoMan Game Competition,” in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, Romania: IEEE, Sep. 2020, pp. 231–237. doi: 10.1109/SYNASC51798.2020.00045.

[68] F. Ishikawa, L. Z. Trovoes, L. Carmo, F. O. D. Franca, and D. G. Fantinato, “Playing Mega Man II with Neuroevolution,” in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, Canberra, ACT, Australia: IEEE, Dec. 2020, pp. 2359–2364. doi: 10.1109/SSCI47803.2020.9308303.

[69] J. Schoemaker and K. Miras, “The benefits of credit assignment in noisy video game environments,” in *The 2022 Conference on Artificial Life*, Online: MIT Press, 2022. doi: 10.1162/isal_a_00483.

[70] N. John and J. Gow, “Adversarial Behaviour Debugging in a Two Button Fighting Game,” in *2021 IEEE Conference on Games (CoG)*, Copenhagen, Denmark: IEEE, Aug. 2021, pp. 01–08. doi: 10.1109/CoG52621.2021.9618893.

[71] B. Maresso, “Emergent behavior in neuroevolved agents.” 2018.

[72] A. Knowles, “Applying neuroevolutionary algorithms to video game artificial intelligence agents,” 2018.

[73] D. Jallov, S. Risi, and J. Togelius, “EvoCommander: A Novel Game Based on Evolving and Switching Between Artificial Brains,” *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 2, pp. 181–191, Jun. 2017, doi: 10.1109/TCIAIG.2016.2535416.

[74] V. Cerny, “Procedural generation of endless runner type of video games.” 2018.

[75] A. Friedman and J. Schrum, “Desirable Behaviors for Companion Bots in First-Person Shooters,” in *2019 IEEE Conference on Games (CoG)*, London, United Kingdom: IEEE, Aug. 2019, pp. 1–8. doi: 10.1109/CIG.2019.8848036.

[76] R. Dubey, S. Louis, A. Gajurel, and S. Liu, “Comparing Three Approaches to Micro in RTS Games,” in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand: IEEE, Jun. 2019, pp. 777–784. doi: 10.1109/CEC.2019.8790308.

[77] A. Gajurel, “Neuroevolution for Realtime Strategy Game Micromanagement,” May 2019.

[78] S. D. Riccio, D. Dyankov, G. Jansen, G. Di Fatta, and G. Nicosia, “Pareto Multi-task Deep Learning,” in *Artificial Neural Networks and Machine Learning – ICANN 2020*, vol. 12397, I. Farkaš, P. Masulli, and S. Wermter, Eds., in Lecture Notes in Computer Science, vol. 12397. , Cham: Springer International Publishing, 2020, pp. 132–141. doi: 10.1007/978-3-030-61616-8_11.

[79] M. L. Clei and P. Bellec, “Generative Adversarial Neuroevolution for Control Behaviour Imitation.” arXiv, Apr. 03, 2023. Accessed: Oct. 17, 2023. [Online]. Available: http://arxiv.org/abs/2304.12432

[80] P. Feldmeier and G. Fraser, “Neuroevolution-Based Generation of Tests and Oracles for Games,” in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, Oct. 2022, pp. 1–13. doi: 10.1145/3551349.3556939.

[81] M. Beukman, C. W. Cleghorn, and S. James, “Procedural content generation using neuroevolution and novelty search for diverse video game levels,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Boston Massachusetts: ACM, Jul. 2022, pp. 1028–1037. doi: 10.1145/3512290.3528701.

[82] P. G. De Prado Salas and S. Risi, “Collaborative interactive evolution in minecraft,” in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Kyoto Japan: ACM, Jul. 2018, pp. 127–128. doi: 10.1145/3205651.3205723.

[83] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. Stanley, and J. Clune, “Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents,” 2018.

[84] E. C. Jackson and M. Daley, “Novelty Search for Deep Reinforcement Learning Policy Network Weights by Action Sequence Edit Metric Distance.” arXiv, Feb. 08, 2019. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/1902.03142

[85] “Sample Efficient Deep Neuroevolution in Low Dimensional Latent Space.” 2019.

[86] S. Khamesian and H. Malek, “Hybrid self-attention NEAT: a novel evolutionary self-attention approach to improve the NEAT algorithm in high dimensional inputs,” *Evolving Systems*, Jun. 2023, doi: 10.1007/s12530-023-09510-3.

[87] Z. Chen, Y. Zhou, X. He, and S. Jiang, “A Restart-based Rank-1 Evolution Strategy for Reinforcement Learning,” in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Macao, China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, pp. 2130–2136. doi: 10.24963/ijcai.2019/295.

[88] G. Cuccu, J. Togelius, and P. Cudré-Mauroux, “Playing Atari with few neurons: Improving the efficacy of reinforcement learning by decoupling feature extraction and decision making,” *Auton Agent Multi-Agent Syst*, vol. 35, no. 2, p. 17, Oct. 2021, doi: 10.1007/s10458-021-09497-8.

[89] P. Templier, E. Rachelson, and D. G. Wilson, “A geometric encoding for neural network evolution,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Lille France: ACM, Jun. 2021, pp. 919–927. doi: 10.1145/3449639.3459361.

[90] A. Asseman, N. Antoine, and A. S. Ozcan, “Accelerating Deep Neuroevolution on Distributed FPGAs for Reinforcement Learning Problems,” *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 2, pp. 1–17, Apr. 2021, doi: 10.1145/3425500.

[91] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning.” arXiv, Apr. 20, 2018. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/1712.06567

[92] M. Shakhova and A. Zagarskikh, “Dynamic Difficulty Adjustment with a simplification ability using neuroevolution,” *Procedia Computer Science*, vol. 156, pp. 395–403, 2019, doi: 10.1016/j.procs.2019.08.219.

[93] N. Beume, B. Naujoks, and M. Emmerich, “SMS-EMOA: Multiobjective selection based on dominated hypervolume,” *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, Sep. 2007, doi: 10.1016/j.ejor.2006.08.008.

[94] Y. Peng, “Policy Direct Search for Effective Reinforcement Learning,” 2019.

[95] D. Perez-Liebana, M. Sajid Alam, and R. D. Gaina, “Rolling Horizon NEAT for General Video Game Playing,” in *2020 IEEE Conference on Games (CoG)*, Osaka, Japan: IEEE, Aug. 2020, pp. 375–382. doi: 10.1109/CoG47356.2020.9231606.

[96] H. Zhang, C.-H. H. Yang, H. Zenil, N. A. Kiani, Y. Shen, and J. N. Tegner, “Evolving Neural Networks through a Reverse Encoding Tree,” in *2020 IEEE Congress on Evolutionary Computation (CEC)*, Glasgow, United Kingdom: IEEE, Jul. 2020, pp. 1–10. doi: 10.1109/CEC48606.2020.9185648.

[97] A. Dharna, A. K. Hoover, J. Togelius, and L. B. Soros, “Transfer Dynamics in Emergent Evolutionary Curricula.” arXiv, Mar. 03, 2022. Accessed: Oct. 25, 2023. [Online]. Available: http://arxiv.org/abs/2203.10941

[98] R. Krauss, M. Merten, M. Bockholt, and R. Drechsler, “ALF -- A Fitness-Based Artificial Life Form for Evolving Large-Scale Neural Networks,” in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Jul. 2021, pp. 225–226. doi: 10.1145/3449726.3459545.

[99] M. Merten, R. Krauss, and R. Drechsler, “Scalable Neuroevolution of Ensemble Learners,” in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, Lisbon Portugal: ACM, Jul. 2023, pp. 667–670. doi: 10.1145/3583133.3590711.

[100] M. Sokół and M. Smołka, “Application of the Hierarchic Memetic Strategy HMS in Neuroevolution,” in *Computational Science – ICCS 2022*, vol. 13351, D. Groen, C. De Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, Eds., in Lecture Notes in Computer Science, vol. 13351. , Cham: Springer International Publishing, 2022, pp. 422–429. doi: 10.1007/978-3-031-08754-7_49.

[101] G. Cui *et al.*, “Reinforced Evolutionary Algorithms for Game Difficulty Control,” in *2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, Sanya China: ACM, Dec. 2020, pp. 1–7. doi: 10.1145/3446132.3446165.

[102] M. Le Cleï, “Evolving artificial neural networks to imitate human behaviour in Shinobi III return of the Ninja master.” Aug. 2021.

[103] A. P. Poulsen, M. Thorhauge, M. H. Funch, and S. Risi, “DLNE: A hybridization of deep learning and neuroevolution for visual control,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, New York, NY, USA: IEEE, Aug. 2017, pp. 256–263. doi: 10.1109/CIG.2017.8080444.

[104] G. Brockman *et al.*, “OpenAI Gym.” arXiv, Jun. 05, 2016. Accessed: Nov. 24, 2023. [Online]. Available: http://arxiv.org/abs/1606.01540

[105] V. Asha, A. Prasad, C. R. Vishwanath, K. Madava Raj, A. R. Manoj Kumar, and N. Leelavathi, “Designing A Popular Game Framework Using Neat A Genetic Algorithms,” in *2023 International Conference on Artificial Intelligence and Applications (ICAIA) Alliance Technology Conference (ATCON-1)*, Bangalore, India: IEEE, Apr. 2023, pp. 1–5. doi: 10.1109/ICAIA57370.2023.10169555.

[106] A. Brandão, P. Pires, and P. Georgieva, “Reinforcement Learning and Neuroevolution in Flappy Bird Game,” in *Pattern Recognition and Image Analysis*, vol. 11867, A. Morales, J. Fierrez, J. S. Sánchez, and B. Ribeiro, Eds., in Lecture Notes in Computer Science, vol. 11867. , Cham: Springer International Publishing, 2019, pp. 225–236. doi: 10.1007/978-3-030-31332-6_20.

[107] C.-B. Pătraşcu and D.-T. Iancu, “Neat algorithm for simple games,” in *2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Bucharest, Romania: IEEE, Jun. 2023, pp. 1–6. doi: 10.1109/ECAI58194.2023.10193858.

[108] A. Darii *et al.*, “Analysis, Combination and Integration of Neuroevolution and Backpropagation Algorithms for Gaming Environment,” in *2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Bucharest, Romania: IEEE, Jun. 2023, pp. 1–5. doi: 10.1109/ECAI58194.2023.10193958.

[109] N. Deepkumar and V. Vasudevan, “Neuro-Evolution in Flappy Bird Game,” in *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)*, Mysuru, India: IEEE, Oct. 2022, pp. 1–5. doi: 10.1109/MysuruCon55714.2022.9972637.

[110] N. Ghebre, E. Jen, M. McBrien, A. Shah, and N. Solomon, “Flappy Bird: Neuroevolution vs NEAT,” 2017.

[111] S. Kumar, G. Khatri, G. Singh, and H. Gupta, “Artificial Intelligent Player Character Using Neuroevolution of Augmenting Topologies and Neural Networks,” in *2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, Chennai, India: IEEE, Jan. 2022, pp. 1–6. doi: 10.1109/ACCAI53970.2022.9752553.

[112] Y. Mishra, V. Kumawat, and K. Selvakumar, “Performance Analysis of Flappy Bird Playing Agent Using Neural Network and Genetic Algorithm,” in *Information, Communication and Computing Technology*, vol. 1025, A. B. Gani, P. K. Das, L. Kharb, and D. Chahal, Eds., in Communications in Computer and Information Science, vol. 1025. , Singapore: Springer Singapore, 2019, pp. 253–265. doi: 10.1007/978-981-15-1384-8_21.

[113] T. Ramaswamy, Y. Pranati, C. V. Sindhu Lahari, and S. Sanjana, “Teaching AI to Play Games using Neuroevolution of Augmenting Topologies,” *International Journal of Innovative Science and Research Technology (IJISRT)*, vol. 7, no. 3, Mar. 2022.

[114] J. P. Selvan and P. S. Game, “Playing a 2D Game Indefinitely using NEAT and Reinforcement Learning.” arXiv, Jul. 28, 2022. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/2207.14140

[115] O. S. Sumukh, P. Fernandes, and M. Meleet, “FLAPPY BIRD AUTOMATION USING REINFORCEMENT ALGORITHM,” *EPRA*, Jul. 2022.

[116] L. Thurler, J. Montes, R. Veloso, A. Paes, and E. Clua, “AI Game Agents Based on Evolutionary Search and (Deep) Reinforcement Learning: A Practical Analysis with Flappy Bird,” in *Entertainment Computing – ICEC 2021*, vol. 13056, J. Baalsrud Hauge, J. C. S. Cardoso, L. Roque, and P. A. Gonzalez-Calero, Eds., in Lecture Notes in Computer Science, vol. 13056. , Cham: Springer International Publishing, 2021, pp. 196–208. doi: 10.1007/978-3-030-89394-1_15.

[117] I. K. Tejani, “Application of Neuroevolution in Blackjack,” Dec. 2020.

[118] S. Pham, K. Zhang, T. Phan, J. Ding, and C. Dancy, “Playing SNES Games With NeuroEvolution of Augmenting Topologies,” *AAAI*, vol. 32, no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.12199.

[119] Dr. M. L. P. Mrs. Smita Premanand, Mr. Kapil Kelkar, “Efficient Application of Competitive Multiagent Learning by Neuroevolution,” *MSEA*, vol. 71, no. 3s, Jul. 2022, doi: 10.17762/msea.v71i3s.24.

[120] M. Sharma, T. Singhal, T. Umar, U. R. Sharma, and V. K. Maurya, “Car Racing Game AI Using Reinforcement Learning,” vol. 02, 2023.

[121] M. Andersson, “How does the performance of NEAT compare to Reinforcement Learning?,” Feb. 2022.

[122] E. Bytyci, A. Hulaj, and D. Aliu, “Combination of Genetic Algorithm and Neural Network for Independent Game Playing,” in *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, Valencia, Spain: IEEE, Oct. 2020, pp. 106–109. doi: 10.1109/IDSTA50958.2020.9264133.

[123] S. Smits, T. Haije, T. Bergmans, and K. Standvoss, “From Genes to Behavior: Solving Super Mario Bros — Twice”.

[124] V. R. Rajalingam and S. Samothrakis, “Neuroevolution Strategies for Word Embedding Adaptation in Text Adventure Games,” in *2019 IEEE Conference on Games (CoG)*, London, United Kingdom: IEEE, Aug. 2019, pp. 1–8. doi: 10.1109/CIG.2019.8847952.

[125] B. D. Bryant and R. Miikkulainen, “A Neuroevolutionary Approach to Adaptive Multi-agent Teams,” in *Foundations of Trusted Autonomy*, vol. 117, H. A. Abbass, J. Scholz, and D. J. Reid, Eds., in Studies in Systems, Decision and Control, vol. 117. , Cham: Springer International Publishing, 2018, pp. 87–115. doi: 10.1007/978-3-319-64816-3_5.

[126] M. Miller, M. Washburn, and F. Khosmood, “Evolving unsupervised neural networks for Slither.io,” in *Proceedings of the 14th International Conference on the*

Foundations of Digital Games, San Luis Obispo California USA: ACM, Aug. 2019, pp. 1–5. doi: 10.1145/3337722.3341837.

[127] W. Price and J. Schrum, "Neuroevolution of Multimodal Ms. Pac-Man Controllers Under Partially Observable Conditions," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand: IEEE, Jun. 2019, pp. 466–473. doi: 10.1109/CEC.2019.8790278.

[128] M. Weeks and D. Binnion, "Training a Neural Network Controlled Non-playing Character with Previous Output Awareness," in *Proceedings of the 2019 ACM Southeast Conference*, Kennesaw GA USA: ACM, Apr. 2019, pp. 202–205. doi: 10.1145/3299815.3314459.

[129] M. Weeks, D. Binnion, A. C. Randall, and V. Patel, "Adventure Game with a Neural Network Controlled Non-playing Character," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Cancun, Mexico: IEEE, Dec. 2017, pp. 396–401. doi: 10.1109/ICMLA.2017.0-129.

[130] M. Weeks, A. K. Chase Randall, and V. Patel, "Neuro-Evolution and Robustness: A Case Study," in *2018 IEEE Games, Entertainment, Media Conference (GEM)*, Galway: IEEE, Aug. 2018, pp. 390–395. doi: 10.1109/GEM.2018.8516464.

[131] L. H. Phuc, K. Naoto, and I. Kokolo, "Learning human-like behaviors using neuroevolution with statistical penalties," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, New York, NY, USA: IEEE, Aug. 2017, pp. 207–214. doi: 10.1109/CIG.2017.8080437.

[132] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-Time Neuroevolution in the NERO Video Game," *IEEE Trans. Evol. Computat.*, vol. 9, no. 6, pp. 653–668, Dec. 2005, doi: 10.1109/TEVC.2005.856210.

[133] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the Galactic Arms Race video game," in *2009 IEEE Symposium on Computational Intelligence and Games*, Milano, Italy: IEEE, Sep. 2009, pp. 241–248. doi: 10.1109/CIG.2009.5286468.

[134] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley, "Petalz: Search-Based Procedural Content Generation for the Casual Gamer," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 3, pp. 244–255, Sep. 2016, doi: 10.1109/TCIAIG.2015.2416206.

[135] R. Singh, N. Mehra, and A. Dhamija, "Natural Selection Simulator using Machine Learning," in *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, Sonepat, India: IEEE, Jul. 2022, pp. 257–261. doi: 10.1109/CCiCT56684.2022.00055.

[136] E. Papavasileiou, J. Cornelis, and B. Jansen, "A Systematic Literature Review of the Successors of 'NeuroEvolution of Augmenting Topologies,'" *Evolutionary Computation*, vol. 29, no. 1, pp. 1–73, Mar. 2021, doi: 10.1162/evco_a_00282.

[137] A. Seth, A. Nikou, and M. Daoutis, "Distributed species-based genetic algorithm for reinforcement learning problems," Apr. 2021.

[138] A. Zollanvari, “Ensemble Learning,” in *Machine Learning with Python*, Cham: Springer International Publishing, 2023, pp. 209–236. doi: 10.1007/978-3-031-33342-2_8.

[139] J. Lehman and K. O. Stanley, “Novelty Search and the Problem with Objectives,” in *Genetic Programming Theory and Practice IX*, R. Riolo, E. Vladislavleva, and J. H. Moore, Eds., in Genetic and Evolutionary Computation. , New York, NY: Springer New York, 2011, pp. 37–56. doi: 10.1007/978-1-4614-1770-5_3.

[140] OpenAI *et al.*, “Dota 2 with Large Scale Deep Reinforcement Learning.” arXiv, Dec. 13, 2019. Accessed: Mar. 03, 2024. [Online]. Available: http://arxiv.org/abs/1912.06680

Appendix

The following table reports the full outcome of the data extraction process (PCG = Procedural content generation; PfE = Play for experience; PtW = Play to win; PM = Player modeling; DAS = direct action selection; IAS = indirect action selection; CF = content features; FA = function approximator; GS = game state; VF = visual frame).

Table A.1. Full data extraction.

Study	L1	L2	Output	Input	ANN	Fitness	EA
[74]	PCG	Game design	DAS	GS	FNN	In-game performance Guidance	GA
[32]	PCG	Game design	CF	GS	FNN	In-game performance Guidance	NEAT-based
[81]	PCG	Levels	CF	Coords	CPPN	Content features Novelty	NEAT-based
[31]	PCG	Levels	CF	Coords	CPPN	Content features Multi-objective Novelty	NEAT-based
[47]	PCG	Levels	CF	Coords	CPPN	Novelty	NEAT-based
[32]	PCG	Levels	DAS	GS	CPPN	Content features Multi-objective Novelty	NEAT-based
[48]	PCG	Levels	CF	Coords	CPPN	Content features Novelty	NEAT-based
[34]	PCG	Levels	CF	Coords	RNN	In-game performance Content features	ES
[59]	PCG	Levels	CF	Coords	CPPN	Content features Multi-objective Novelty	NEAT-based
[34]	PCG	NPCs	DAS	GS	CNN	In-game performance	
[39]	PCG	NPCs	DAS	GS	DNN	In-game performance Guidance	NEAT-based
[36]	PCG	NPCs	DAS	GS	RNN	In-game performance	NEAT-based
[37]	PCG	NPCs	DAS	GS	RNN	In-game performance	NEAT-based
[30]	PCG	Other	CF	Coords	CPPN	Interactive	NEAT-based
[28]	PfE	Automatic playtesting	DAS	GS	FNN	Off-game performance	NEAT-based
[29]	PfE	Automatic playtesting	DAS	GS	FNN	Off-game performance	NEAT-based
[80]	PfE	Automatic playtesting	DAS	GS	FNN	Off-game performance	NEAT-based

Table A.1. Continued.

Study	L1	L2	Output	Input	ANN	Fitness	EA
[70]	PfE	Automatic playtesting	DAS	GS	FNN	In-game performance Guidance	NEAT-based
[60]	PfE	Automatic playtesting	DAS	VF	DNN	Off-game performance Multi-objective Guidance	MOEA
[101]	PfE	Difficulty control	DAS	VF	DNN	Relative performance Guidance	REA-DD/RMOEA-DD
[71]	PfE	Difficulty control	DAS	GS	FNN	In-game performance Guidance	GA
[27]	PfE	Difficulty control	IAS	GS	FNN	Relative performance Guidance	GA
[92]	PfE	Difficulty control	DAS	GS	FNN	Relative performance Guidance	GA
[7]	PfE	Difficulty control	IAS	GS	FNN	Relative performance Guidance	NEAT-based
[75]	PfE	NPC behavior	DAS	GS	FNN	In-game performance Multi-objective	NEAT-based
[73]	PfE	NPC behavior	DAS	GS	FNN	In-game performance Guidance	NEAT-based
[72]	PfE	NPC behavior	DAS	GS	FNN	In-game performance Guidance	NEAT-based
[135]	PfE	NPC behavior	DAS	GS	FNN	In-game performance Guidance	NEAT-based
[88]	PtW	ATARI	DAS	VF	RNN	In-game performance	ES
[90]	PtW	ATARI	DAS	VF	DNN	In-game performance	GA
[62]	PtW	ATARI	DAS	VF	HN	In-game performance	GA
[87]	PtW	ATARI	DAS	VF	DNN	In-game performance	ES
[83]	PtW	ATARI	DAS	RAM	DNN	In-game performance Novelty	ES
[8]	PtW	ATARI	DAS	VF	DNN	In-game performance	ES
[84]	PtW	ATARI	DAS	VF	DNN	In-game performance Novelty	GA
[86]	PtW	ATARI	DAS	VF	FNN	In-game performance	ES NEAT-based
[49]	PtW	ATARI	DAS	VF GS	FNN	In-game performance	ES NEAT-based
[61]	PtW	ATARI	DAS	VF	DNN	Relative In-game performance	GA ES
[94]	PtW	ATARI	DAS	RAM	FNN	In-game performance	NEAT-based

Table A.1. Continued.

Study	L1	L2	Output	Input	ANN	Fitness	EA
[78]	PtW	ATARI	DAS	VF	DNN	In-game performance Multi-objective Guidance	ES
[45]	PtW	ATARI	DAS	VF	DNN	In-game performance	ES
[137]	PtW	ATARI	DAS	VF	DNN	In-game performance	GA
[100]	PtW	ATARI	DAS	VF	DNN	In-game performance	ES
[91]	PtW	ATARI	DAS	VF	DNN	In-game performance Novelty	GA
[89]	PtW	ATARI	DAS	RAM	DNN	In-game performance	ES
[42]	PtW	ATARI	DAS	RAM	RNN	In-game performance	NEAT-based
[46]	PtW	ATARI	DAS	VF	RNN	In-game performance	NEAT-based
[85]	PtW	ATARI	DAS	VF	RNN	In-game performance	ES
[57]	PtW	ATARI	DAS	GS	FNN	In-game performance	NEAT-based
[24]	PtW	Board game	DAS	GS	DNN	In-game performance	ES
[26]	PtW	Board game	IAS	GS	FNN	In-game performance	NEAT-based
[25]	PtW	Board game	DAS	GS	FNN	In-game performance	NEAT-based
[20]	PtW	FightingICE	DAS	GS	FNN	In-game performance Multi-objective Guidance	NEAT-based
[21]	PtW	FightingICE	DAS	GS	DNN	Off-game performance	CA
[22]	PtW	FightingICE Racing	DAS	GS	FNN	In-game performance	NEAT-based
[107]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[51]	PtW	Flappy Bird	DAS	GS	LSTM FNN	In-game performance	NEAT-based
[105]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[106]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	GA
[63]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance Guidance	NEAT-based
[108]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NE
[109]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	GA
[110]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[15]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[111]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[112]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	GA
[113]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[114]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based

Table A.1. Continued.

Study	L1	L2	Output	Input	ANN	Fitness	EA
[115]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[116]	PtW	Flappy Bird	DAS	GS	FNN	In-game performance	NEAT-based
[17]	PtW	FPS	DAS	VF	DNN	In-game performance	ES
[9]	PtW	FPS	IAS	GS	FNN	In-game performance	NEAT-based
[103]	PtW	FPS	DAS	GS	FNN	In-game performance	NEAT-based
[56]	PtW	FPS Racing	DAS	VF	LSTM	In-game performance	ES
[52]	PtW	IIG	DAS	GS	LSTM FNN	In-game performance	GA
[117]	PtW	IIG	DAS	GS	FNN	In-game performance	NEAT-based
[23]	PtW	IIG	DAS	GS	LSTM FNN	In-game performance	GA
[53]	PtW	IIG	DAS	GS	LSTM FNN	In-game performance	ES
[67]	PtW	EvoMan	DAS	GS	FNN	In-game performance	GA
[68]	PtW	EvoMan	DAS	GS	FNN	In-game performance Guidance	GA
[69]	PtW	EvoMan	DAS	GS	FNN	In-game performance	GA NEAT-based
[44]	PtW	NES/SNES	DAS	RAM	FNN	In-game performance Guidance	NEAT-based
[118]	PtW	NES/SNES	DAS	VF	FNN	In-game performance	NEAT-based
[43]	PtW	NES/SNES	DAS	RAM	RNN	In-game performance	NEAT-based
[95]	PtW	Other	DAS	GS	FNN	In-game performance	NEAT-based
[124]	PtW	Other	DAS	GS	FNN	In-game performance	ES
[97]	PtW	Other	DAS	GS	FNN	In-game performance	PYNSKY
[125]	PtW	Other	DAS	GS	FNN	In-game performance	NE
[82]	PtW	Other	DAS	GS	FNN	Interactive	NEAT-based
[126]	PtW	Other	DAS	GS	FNN	In-game performance	NEAT-based
[127]	PtW	Other	DAS	GS	FNN	In-game performance	GA
[128]	PtW	Other	DAS	GS	FNN	In-game performance	GA
[129]	PtW	Other	DAS	GS	FNN	In-game performance	GA
[130]	PtW	Other	DAS	GS	FNN	In-game performance	GA
[96]	PtW	Other	DAS	GS	FNN	In-game performance	NEAT-based
[13]	PtW	Pong	DAS	VF GS	DNN	In-game performance	GA
[66]	PtW	Pong	DAS	GS	FNN	In-game performance Guidance	NEAT-based

Table A.1. Continued.

Study	L1	L2	Output	Input	ANN	Fitness	EA
[14]	PtW	Pong	DAS	GS	FNN	In-game performance	NEAT-based
[119]	PtW	Pong	DAS	GS	FNN	In-game performance	NEAT-based
[55]	PtW	Racing	DAS	VF	DNN LSTM	In-game performance	GA
[120]	PtW	Racing	DAS	GS	FNN	In-game performance	GA
[18]	PtW	RTS	DAS	GS	FNN	Relative In-game performance Guidance	NEAT-based
[76]	PtW	RTS	DAS	GS	FNN	In-game performance Multi-objective	GA NEAT-based
[77]	PtW	RTS	DAS	GS	FNN	In-game performance Multi-objective	GA NEAT-based
[99]	PtW	SMB	DAS	GS	FNN	In-game performance	TWEANN
[121]	PtW	SMB	DAS	GS	FNN	In-game performance	NEAT-based
[122]	PtW	SMB	DAS	GS	FNN	In-game performance	GA
[64]	PtW	SMB	DAS	GS	FNN	In-game performance Guidance	NEAT-based
[65]	PtW	SMB	DAS	GS	FNN	In-game performance Guidance	NEAT-based
[98]	PtW	SMB	DAS	GS	FNN	In-game performance	ALF
[123]	PtW	SMB	DAS	VF	FNN	In-game performance	NEAT-based
[50]	PtW	Tetris	DAS	VF GS	CPPN	In-game performance Multi-objective	GA NEAT-based
[58]	PtW	Tetris	DAS	VF	CNN	In-game performance Multi-objective	GA NEAT-based
[102]	PM		DAS	VF	CNN RNN	Off-game performance Relative performance	NEAT-based
[79]	PM		DAS	GS	RNN	Off-game performance Relative performance	GA
[54]	PM		DAS	GS	LSTM	In-game performance	NEAT-based
[131]	PM		DAS	GS	FNN	In-game performance Off-game performance	NEAT-based
[38]	PM		FA	VF	FNN	Off-game performance	NEAT-based

The following table gathers additional tags used in data extraction.

Table A.2. Additional tags

Studies	Tag
[74] [31] [61] [86] [102] [59] [34] [24] [97] [76] [77] [18] [64] [79]	Coevolutionary solution
[32] [30] [75] [72] [73]	Survey-based evaluation
[74] [81] [31] [101] [88] [78] [61] [86] [8] [42] [45] [46] [49] [62] [83] [84] [85] [87] [89] [90] [91] [94] [100] [137] [22] [51] [107] [56] [95] [124] [99] [102]	Multiple games
[88] [32] [47] [97]	Open-endedness