Chapter 25
Evolutionary Machine Learning and Games

Julian Togelius, Ahmed Khalifa, Sam Earle, Michael Cerny Green, and Lisa Soros

Abstract Evolutionary machine learning (EML) has been applied to games in multiple ways, and for multiple different purposes. Importantly, AI research in games is not only about playing games; it is also about generating game content, modeling players, and many other applications. Many of these applications pose interesting problems for EML. We will structure this chapter on EML for games based on whether evolution is used to augment machine learning (ML) or ML is used to augment evolution. For completeness, we also briefly discuss the usage of ML and evolution separately in games.

25.1 Introduction

Games of all sorts (including card games, board games, and video games) provide a rich domain for exploring computational intelligence. In many ways, games reflect the parts of the real world that we as humans find interesting, isolating key facets of our experience and encapsulating them within a tractable and interactive medium.

Beyond mere entertainment, games provide a unique domain for exploring and evaluating AI. Historically, the intersection of AI and games has focused on agents that play specific games *well*. In this way, games complement the litany of task environments for AI such as embodied agent control. However, games offer additional

J. Togelius (✉) · S. Earle · M. C. Green
Computer Science and Engineering, New York University, 370 Jay Street, Brooklyn, NY 07960, USA
e-mail: julian.togelius@gmail.com

J. Togelius
Odl.ai, Nørrebrogade 184, 2200 Copenhagen, Denmark

A. Khalifa
Institute of Digital Games, University of Malta, 20 TriqL-Esperanto, Msida MSD2080, Malta

L. Soros
Computer Science, Barnard College, 3009 Broadway, New York NY 10027, USA

W. Banzhaf et al. (eds.), *Handbook of Evolutionary Machine Learning*,
Genetic and Evolutionary Computation,
https://doi.org/10.1007/978-981-99-3814-8_25

challenges beyond reward maximization such as effecting spirited play or emulating the style of particular humans.

In addition to playing games, there are challenges including generating content (such as levels, quests, textures, and characters), modeling players, matching players, and adapting interfaces. Content generation in particular requires deeply creative algorithms capable of understanding the essence of a domain and conjuring up new artifacts. In this way, games also provide an opportunity for exploring concepts such as algorithmic innovation and open-endedness.

In this chapter, we survey the application of EML to games. We take an inclusive view of EML, focusing on cases where a ML model is evolved, but including examples of all kinds of interaction between evolution and ML. We also give brief overviews of the use of non-evolutionary ML and non-ML evolution in games, but given the breadth of the topic, those sections are mere sketches.

EML can, in one form or another, be applied to almost any AI challenge in games. However, this family of methods has seen much more application in some areas rather than others. Reflecting on this, a relatively large number of examples in this chapter will be from game content generation. But there will also be plenty of examples of game-playing evolutionary ML.

25.2 Machine Learning in Games

Some of the earliest advancements in ML are due to research on game-playing. In particular, Samuel's Checkers player from 1959 [81] was the first example of what we now call reinforcement learning. While programs for Chess [12], Checkers [82], and Go initially built (and usually still build) on some form of tree search, machine-learned board value functions were introduced at an early stage and became crucial to any advanced efforts to play classical board games. These value functions could be learned through supervised learning, reinforcement learning, or some combination. AlphaGo [89] and AlphaZero [90] represent very successful combinations of tree search with ML that originated in research on Go playing but have found applications in a wide variety of fields.

At the outset of the deep learning era, certain video games came to play important roles as testbeds and benchmarks of deep reinforcement learning algorithms. Deep Q-networks were introduced in a landmark Nature paper in 2015 [67], where they were trained on games from the Atari 2600 console via the Arcade Learning Environment [6]. Since then, Atari games have been a crucial benchmark for deep reinforcement learning, and new algorithms are often tested on Atari games first [49]. Several other video games have also become important as benchmarks for RL algorithms, notably Doom [52] and MineCraft [40]. To a lesser extent, supervised learning has been applied to these games to learn game-playing from large sets of human playtraces [3].

Although a much smaller research area, ML algorithms of various kinds have also been applied to generating game content [100]. Most of these applications fall into self-supervised learning, such as using generative adversarial networks or recurrent neural networks for generating levels for Super Mario Bros [2, 99, 112] or cards for Magic: The Gathering [98]. However, there have also been attempts to use reinforcement learning for level generation more recently [53].

Other AI challenges in games for which ML has been used prominently include player modeling [91], cheat detection [47], and matchmaking [66]. In player modeling, preference learning has been successfully applied to predicting player affect in response to game levels, game situations, or behavior. For cheat detection, explainable ML algorithms (such as decision trees) are most commonly used as the AI cannot act by itself and block players. It usually needs the moderator's input to confirm if the flagged player is a cheater before banning the player from the game. Microsoft's TrueMatch system, widely used in XBox games, uses ML on top of a ranking system to learn to match players well.

25.3 Evolution in Games

The robustness and wide applicability of evolutionary search means that evolutionary algorithms have been applied rather extensively to games. Many, perhaps most, examples of applications of evolution to games are in the form of neuroevolution, which counts as EML. However, there are also "pure" applications of evolutionary computation in games. Here we will survey two types: evolutionary planning and search-based procedural content generation.

25.3.1 Evolutionary Planning

For playing a game that has a fast-forward model, one preferably uses some kind of search algorithm for planning. Traditionally, this would mean a tree search algorithm, such as a Minimax variation for two-player games or some form of A* for single-player games [79]. For games with a nontrivial search depth, one would need to cut off the search at some specified depth and use a board or state evaluation function to evaluate the deepest search node.

The plan that is generated by the tree search algorithm is simply a list of actions, with a numeric value (obtained from the state evaluation function) that signifies the projected value of that sequence of actions. Recognizing this, one might just as well use an evolutionary algorithm to evolve the plan, using the state evaluation function as a fitness function. In an approach called rolling horizon evolution [71], several groups have done this for single-player games [31], with results that are generally competitive with state-of-the-art tree search algorithms such as Monte Carlo Tree Search [11].

The advantages of evolution over classic tree search algorithms become more clear in games with a very large branching factor [84]. Games that feature control over multiple units, including many real-time or turn-based strategy games, can have branching factors in millions or even billions. This chokes most tree search algorithms as the search never gets past the first few moves. Evolutionary algorithms, operating on the level of the whole plan as a sequence, are not affected by this limitation. Results for several strategy games [50] show that evolutionary planning can out-compete tree search by a large margin.

25.3.2 Search-Based PCG

After game playing, procedural content generation (PCG) is probably the topic in games that has received the most interest from the AI research community [86]. Within academic research on PCG, evolutionary computation is currently the dominant approach because of the natural fit between the method and the problem. In search-based PCG, as the application of evolution to PCG problem is called, individual pieces of game content are evolved with a fitness function that reflects some measure of content quality [109]. This approach has been applied to a large number of types of game content, including quests [20], music [26], textures [114], rules [17], and levels for platform games [85], first-person shooters [13], and real-time strategy games [62].

Finding a fitness function that accurately reflects the quality of a candidate content artifact is typically nontrivial. Judging the quality of, for example, a game level is inherently hard and often requires playing the game. Writing code that performs this quality judgment is often harder still. Therefore, many search-based PCG implementations use simulation-based evaluation, where an agent of some kind plays part of the game that incorporates the content under evaluation. For example, the agent could play a level to see if it is playable under certain conditions [54]. Such agents may be based on ML [108]. In other cases, the fitness function is model-based and relies on some kind of learned model to do the evaluation [51]. In yet other works, fitness is assigned interactively by a human player [32].

25.4 Evolutionary Machine Learning in Games

In this section, we overview the use of EML in games research. While one way of categorizing this body of work is to divide game application (e.g. planning for non-player character actions, and PCG), another is to tear apart EML, considering the ways in which evolution and ML interact. We take the latter approach in the context of this book, as the former is better suited to works on AI and Games [116]. We survey research in games using EML (as shown in Fig. 25.1) by

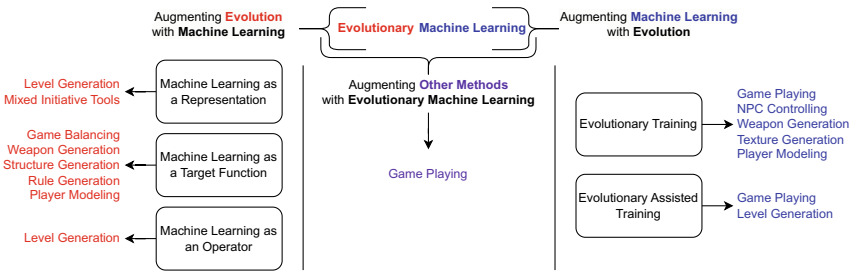


Fig. 25.1 Taxonomy of EML in Games based on the dynamics between evolution and ML with examples of the usage of these techniques in games

- Augmenting Machine Learning with Evolution.
- Augmenting Evolution with Machine Learning.
- Augmenting other methods with Evolutionary Machine Learning.

The way we acquired this division is by looking into what is the end goal of the work that uses EML and analyze what is the core algorithm that is being used. For example, if the end goal is to have a neural network agent that controls a player in a video game, then evolution in that case is helping the ML (neural network) to achieve its goal. On the other hand, if the goal is to use evolution to find a certain content, then the search process is what is important and the ML is helping it to achieve its goal. We also found some other cases where the goal algorithm is neither an evolutionary algorithm nor ML, but where EML nonetheless plays an important supporting role.

25.5 Augmenting Machine Learning with Evolution

The easiest way to incorporate the evolutionary algorithm with ML is to utilize evolutionary optimization power to help the ML algorithm. Thinking about it from this direction allows us to see two ways where evolution can help. We can either use an evolutionary algorithm as the training process for the ML or the evolutionary algorithm for adjusting and helping the training process of ML, or both.

25.5.1 Evolutionary Training

Evolution can be used to directly train neural networks, either the weights of the network, its topology, or both. This is called *neuroevolution* and has a long history in artificial life, robotics [68], and video game research [76]. It should be noted that some recent neuroevolution work, specifically focused on finding structures for deep networks, is alternatively billed as "neural architecture search" [27].

25.5.1.1 Neuroevolution for Gameplay Tasks

The probably most common use for neuroevolution in games is as a form of reinforcement learning. Here, the fitness evaluation of a chromosome is calculated by converting the genotype to a neural network (which might be as simple as setting the weights of a fixed topology) and using it to try to play the game, and basing the fitness on how well the network played the game. Compared to other forms of reinforcement learning, such as those based on variations of temporal difference learning or policy gradients, neuroevolutionary reinforcement learning tends to be more reliable but has issues with scaling to very large networks [76, 107].

In board games, the most common (and probably most sensible) version of neuroevolution is to evolve a board value function used together with some version of the Minimax algorithm. Successful applications of this method go back a few decades [28]; see also Sect. 25.7.

In video games, where fast and deterministic forward models are rare, it is typically more appropriate to use neuroevolution to generate a network that directly selects what action to take based on an observation of the environment. Early examples of this type of neuroevolution include work on 2D car racing games [105, 106]. In those examples, the neural networks were fed simulated rangefinders representing the track geometry in front of the car, and outputted accelerator/brake and steering commands. Another relatively early example of this type of neuroevolution is the evolution of neural nets to play Super Mario Bros based on discrete grid sensor inputs [104].

As the ALE benchmark suite, featuring games from the old Atari 2600 video game console, became popular with the rise of deep reinforcement learning, evolutionary algorithms were also applied to learn to play Atari games from pixel inputs. An influential paper by a team at OpenAI showed that a simple evolution strategy can be competitive with standard DQN, and in particular that evolution scales effortlessly compared to other types of reinforcement learning [80]. Follow-up work by other authors showed that the slightly more sophisticated Canonical Evolution Strategy can perform much better [16]. However, the more recent and sophisticated gradient-based RL methods outperform existing evolutionary methods on Atari games. In general, while standard neuroevolution remains competitive with gradient-based RL when the network size is small, it tends to not scale very well to very large networks. This may be because the single fitness measure imparts less information than the dense rewards that can be used with some RL methods, or because of the lack of directional information in the random mutations in a high-dimensional space.

Various attempts have been made to overcome this limitation of neuroevolutionary reinforcement learning for games with high-dimensional (such as visual) input space. One idea is to separate visual processing from the policy. This way, a smaller network that receives a lower dimensional encoding of the observation can be trained effectively by evolution. One idea is to use an autoencoder to compress the visual input, feeding the smaller policy network of the bottleneck layer in the autoencoder. This was tested successfully on the classic FPS game Doom, in a setup where the autoencoder was continually re-trained as new sections of the level were discovered [1]. Another idea is to represent the state using a dictionary of centroids in state

space, feeding a sparse encoding to the policy network. This setup can be used to evolve surprisingly tiny networks, with as few as six neurons, to play Atari games well [18].

One of the areas where neuroevolution can do things that regular reinforcement arguably cannot is in topology and weight evolution, where both the topology and the weights of the network are evolved. The pre-eminent algorithm here is NEAT by Stanley and Miikkulainen, which has been applied to learn to play various games and performs very well on small to moderate-size networks [96]. Later, it has been applied to the General Video Game framework with decent results [72]. HyperNEAT is an indirect encoding that can in principle handle much larger input spaces [95], and which has been applied to playing Atari games with some success [14].

25.5.1.2 Neuroevolution for Non-gameplay Tasks

Neuroevolution can also be applied to other tasks in games besides game playing. In particular, there are several prominent examples of neuroevolution for procedural content generation. This includes the Galactic Arms Race game [43], where players collaboratively evolve weapons, and the Petalz social network game where players evolve flowers [75]. One can even use neuroevolution as a form of meta-content generator, evolving a population of networks that can generate a large range of different levels [24]. An interesting use of neuroevolution for non-player characters is in the NERO game, where players train populations of agents to fight for them [94]. Even earlier, the cult video game Creatures uses a form of neuroevolution as a game mechanic.

Neuroevolution has also been used to model a certain playstyle [45, 46]. Holmgaard et al. used a $\mu + \lambda$ evolutionary strategy to evolve a single-layer neural network that decides on the next target in MiniDungeons. The agent then uses the A* algorithm to navigate toward the selected target. The agents were evolved using action agreement ratio [45], tactical agreement ratio, and strategic agreement ratio [46] to mimic different playstyles from a corpus of collected human data. The final evolved agents were not only able to finish unseen levels but also navigate them in the same vein as the target playstyle.

25.5.2 Evolutionary Assisted Training

In the work described in Sect. 25.5.1 above, the evolutionary process serves to learn more sophisticated policies—e.g. for game playing or design, whether by the modification of action sequences, game assets, or of neural network weights or architectures. In this section, on the other hand, the learning or training process is implemented by some non-evolutionary algorithm (typically gradient descent), while evolution exists as an auxiliary process that renders learning more capable or efficient. This section focuses primarily on a particularly prominent role taken on by such an

assistive evolutionary process, which is creating or curating a "curriculum" for a learning agent.

25.5.2.1 Curriculum Design in Machine Learning

Curriculum learning [7] takes inspiration from human developmental psychology, where, by way of example, children are taught complex subjects (e.g. mathematics) incrementally over time, by first familiarizing them with more elementary and fundamental concepts (such as basic arithmetic) before moving on to more sophisticated techniques (such as calculus or linear algebra). In the context of ML, a curriculum may expose a learning agent to increasingly complex training examples, switching from simpler to more complex ones once the agent's abilities reach a certain level.

For example, a neural network-based agent tasked with navigating through a maze to a goal tile will in all likelihood be incapable of navigating a complex maze by sheer luck at the very beginning of training (i.e. with randomly initialized weights resulting in it taking effectively random actions at each step). It would thus be wasteful to expose it to such complex mazes at the beginning of training. It has a much better chance, on the other hand, of stumbling upon the goal tile in simpler mazes, where the goal is close to the starting position. A curriculum learning approach could thus order mazes by their complexity (or the distance between start and goal positions) only introducing more complex mazes into the pool of training data once the agent has learned to reliably solve a set of simpler mazes.

In ML, curriculum learning can not only make training more efficient (avoiding training on infeasible examples early on) but also lead to improved generalization, by allowing the model to incrementally learn more meaningful representations of the problem at hand. The importance of curriculum learning for training generally capable strategies over a large space of tasks has been demonstrated in XLand [102, 103], where a large set of tasks are generated in a 3D environment, where embodied agents take visual input and must navigate a procedurally generated terrain to achieve certain goal states, which involve manipulating and re-combining various primitive objects. In XLand, a massive space of tasks (involving different rules and initial map conditions) are generated before learning, then curated during agent training so as to produce a curriculum of increasing complexity. This curation is aided by metrics using agent play-through to measure whether a given task is both nontrivial and learnable (i.e. whether it is on the "frontier" of agent abilities). The resulting player agents are able to generalize to new tasks—involving different object-recombination mechanics and goal states—that were not seen during training.

25.5.2.2 Evolutionary Curriculum Design

Other work generates training environments on the fly, then curates or filters them to ensure learnability, resulting in a series of environments of increasing complexity over the course of agent training player agents. PAIRED [21] introduces this notion

of "Unsupervised Environment Design", using an RL agent to generate level layouts of training environments so as to maximize learnability (formulated as approximate regret). More recently, ACCEL [70] effectively evolves these environments, applying small mutations to level layouts and filtering them for regret. The domains considered include tile-based maze-like environments in which the player must navigate to a goal (while sometimes avoiding lava) and the 2D physics-based "bipedal walker" environment. The mutations involve changing the state of given tiles in the former case and changing the height of the terrain in the latter. The fitness criterion of a mutated level is the regret it induces in the player agent. The authors show that the evolution of increasingly complex environments (lengthy and obstacle-heavy mazes, and rough or steep terrain) coincide with the development of a robust player agent capable of generalizing to unseen environments (e.g. capable of solving a maze in a grid much larger than those evolved during training).

POET [113] similarly evolves environments online while training player policies in the bipedal walker environment, though it trains multiple player agents concurrently, pairing these with particular environments (and occasionally transferring players to different environments mid-training in a paradigm reminiscent of transfer learning [110]). Here, terrain is represented by a Compositional Pattern Producing Network (CPPN [93]), and evolved using Neuroevolution of Augmenting Topologies (NEAT [96]). Agent policies are optimized via an Evolutionary Strategy [80]. Mutated environments are added to the pool of environments if they are neither too hard nor too easy for existing agents, and if they are sufficiently novel with respect to environments already in the pool, thus encouraging diversity among the problems generated (and solved) by the algorithm. PINSKY [22] applies the framework introduced by POET to 2D tile-based games.

In Go-Explore [25], a variant of novelty search [60] is used to incentivize exploration by an RL agent, by effectively maintaining an archive of behaviors that result in novel states in a sparse reward problem. The domain tackled in this work is Montezuma's Revenge, a side-scrolling adventure game from the Atari 2600 suite, which was formerly unsolved by methods not relying on expert demonstrations. Go-Explore facilitates exploration by looking at the (x, y) coordinate location of the player avatar on the screen, and, whenever a new position is reached by the player, storing in memory the sequence of behaviors that led to this novel state. On subsequent play-throughs, the agent can then reliably navigate back to an (x, y) position on the frontier of that which it has already explored, so that it will tend to explore further and further out from its starting location, and so that once a tricky segment has been solved—such as jumping carefully across a dangerous chasm—it can be reliably solved on every subsequent play-through without further trial and error. When the goal state is reached, they train a game-playing agent on the successful trajectories from the archive.

25.6 Augmenting Evolution with Machine Learning

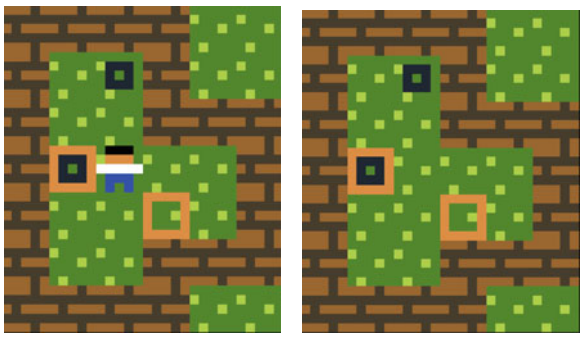
The goal in augmenting the evolutionary process is to mimic the output of the evolution process by using ML to create an easier landscape to navigate, better operators, faster fitness function, etc.

25.6.1 Machine Learning as a Representation

Evolutionary algorithms are sensitive to the input representation [35]. There is a body of research on understanding and exploring the different representations and their effects [78]. In most of these works, it is agreed that a good representation provides us with a smooth fitness landscape. This means that individuals that are close to each other in representation space have similar fitness. In many game domains such as level generation, a direct representation (2D matrix of tiles) is usually not the best as it is too sensitive to small changes. For example, removing the player tile from a level will make the level unplayable immediately (Fig. 25.2). Instead of utilizing domain knowledge to figure out a good representation, ML models can be used to learn this representation. In that case, ML acts as a genotype-to-phenotype mapper for the evolutionary algorithm. The evolutionary algorithm can just work on the provided representation (called latent variables) and modify it using a mutation function, then the ML transforms it to the phenotype where it gets evaluated using a fitness function.

25.6.1.1 Super Mario Bros

Although there are many ML methods that can be used to learn a good representation, most of the work focuses on either AutoEncoders [57] or Generative Adversarial Networks (GANs) [36]. For example, Volz et al. [112] trained a GAN on the levels of Super Mario Bros (Nintendo, 1985). The levels are divided into single scenes of



(a) (b)

Fig. 25.2 An example of a Sokoban level with and without the human player

size (28 × 14) using a sliding window. After training the network on generating new scenes, they used CMA-ES [42] algorithm to search the latent space of the GAN for playable Mario levels with different game features such as maximizing/minimizing the number of jumps. The fitness was calculated using an A* agent that measures the playability percentage of the generated levels and the number of jumps that have been performed. The generated levels mostly follow the structure of original Mario levels without the need for specifying that in the fitness function or the representation [19, 85]. Following the previous work, Fontaine et al. [29] used a quality diversity algorithm called Covariance Matrix Adaptation MAP-Elites (CMA-ME) [30] to discover new and diverse playable levels. Utilizing CMA-ME helped the authors overcome the repetition of the generated levels using normal CMA-ES and find new levels that are hard to find using normal CMA-ES such as levels that combine two different level styles (Fig. 25.3). Another advantage is at the end of the run, the algorithm manages to produce a corpus of playable levels with different features instead of a single playable level as in the case of CMA-ES.

So far, the previously discussed work only focused on generating a small scene of Super Mario levels of size (28 × 14); creating a full level by concatenating the generated scenes sometimes might end in an unplayable level (see Fig. 25.4). This is because evolution does not have any context about the previously generated scenes. This can be easily solved by modifying the fitness to incorporate all the previous scenes. The problem is that evolution cannot change anything from the previously generated scenes. This might cause evolution to repeat certain patterns to solve the problem. Another simple solution is to evolve the whole level all at once and use the GAN to transform each sequence independently. The problem with that is for very long levels; this could be hard as the representation might be big and not easy to evolve due to conflict in fitness between different parts of the chromosome. To solve this issue, Schrum et al. [83] proposed encoding the big level using a compressed representation which is encoded in the form of a CPPN [93]. The CPPN takes the location of the scene, and it spits out the latent variable for that area. The NEAT [96] algorithm was used to create the full level. The output of this experiment shows that

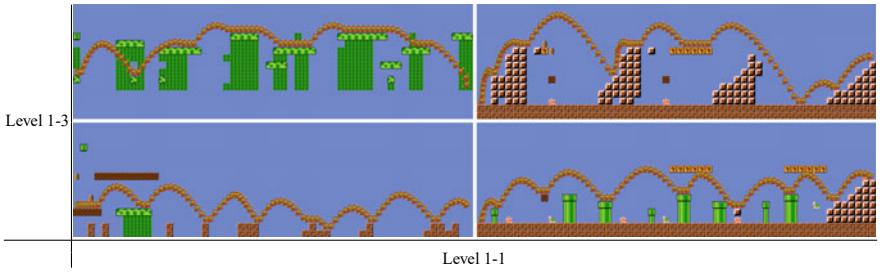


Fig. 25.3 Generated Super Mario scenes using CMA-ME that are similar to either sky levels or ground levels

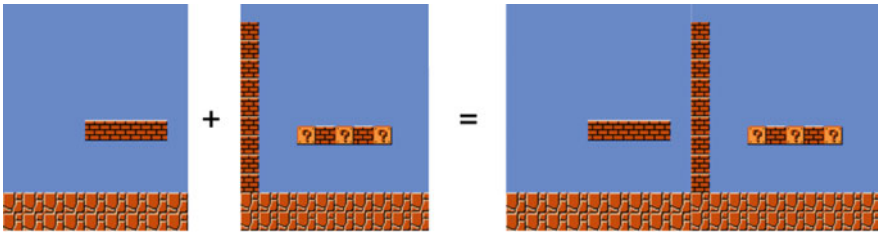


Fig. 25.4 These Mario scenes are playable on their own, but combining them in that order will not be playable

using CPPNs to evolve the structure of the level followed by normal evolution on the concatenated latent variables produced better results than any part alone.

25.6.1.2 Other Applications

Tanabe et al. [101] used latent variable evolution with Variational AutoEncoder (VAE) [74] to generate levels for Angry Birds (Rovio, 2009). They treated the generated levels as a sequence instead of being a tile map to overcome the problem of having variable size objects and having objects overlayed each other. They searched the latent space of VAE using CMA-ES algorithm to find levels that minimize the usage of birds, maximize the number of pigs, etc. search. Another avenue is to use interactive evolution to create a mixed initiative tool [87] combined with the power of ML. This is usually a common avenue in subjective domains that are hard to measure its success. There is a substantial breadth of work in this area, and a full review is outside the scope of this book chapter. However, some recent works generate art [48], human faces [117], backgrounds, and shoes [10]. However, as far as we know, these tools have not been used in texture generation or game art. This could be an interesting avenue to explore with the rise of large generative models such as Stable Diffusion [77].

25.6.2 Machine Learning as a Target Function

Usually, the bottleneck for evolution is calculating the fitness function [92], which motivated people to seek methods to approximate its value to speed up the evolutionary process. This approach involves surrogate models [92]. Surrogate models have been modeled using different ML methods such as linear regression [37], Gaussian mixture models [63], and neural networks [51]. In this section, we decided to call it a target function instead of a fitness function as ML could be used to approximate not only the fitness function but also features of the phenotype such as time to finish the level, and number of jumps in a playtrace. These features can be used either as

behavior characteristics in quality diversity algorithms [8, 118] or a diversity score for divergent search algorithms [5, 61].

We can divide surrogate models based on how they are trained: online and offline methods. The online method focuses on training the surrogate model as a part of the evolutionary process. The evolution helps to create the dataset for training the ML model. On the other hand, the offline method focuses on training the ML model beforehand. This makes surrogate models similar to reinforcement learning where the online method is similar to off-policy reinforcement learning and the offline method is similar to offline reinforcement learning where we collect the data before training the model.

25.6.2.1 Online Methods

In the games domain, most of the work utilizes online training due to the small number of data that can be found to train ML models. Volz et al. [111] used multi-objective optimization to create balanced cards for the Top Trumps card game (Dubreq, 1978). The goal is to generate a group of unique balanced decks such that there is no dominant strategy to win the game. The surrogate model here is a statistical model that predicts the minimum number of simulation that is needed to have an accurate estimation of the win rate. The results showed that generated decks from surrogate models are as good as normal simulations and require a lot less computational power. In a similar vein, Zhang et al. [118] used Deep Surrogate Assisted MAP-Elites to generate a deck of cards for Hearthstone (Blizzard, 2014). They not only used the surrogate model to calculate the fitness (the average difference of health between both players) but also the behavior characteristics (the average number of cards in hand and the number of turns till the end of the match) needed for the MAP-Elites Archive. The model is trained online after a fixed number of iterations to make sure the output is correct and up to date. Bhatt et al. [8] generalized the system by adding another surrogate model that predicts the agent playtrace instead of just the final metrics and tested it on generating mazes and Super Mario Bros levels. The new system works better in comparison with the previous one introduced by Zhang et al. [118] as the introduced agent prediction helps improve the results.

Some quality diversity and divergent search algorithms need ML as their core element such as Surprise Search [37]. Surprise Search abandons objectivity for the sake of surprise. The surprise score needs to be modeled using a ML algorithm so the evolution can predict new elements. The authors modeled the surprise score using linear regression to predict the genotype of the next generation given the previous generation. In this case, online models are being used as the model gets updated after each generation. Later, Gravina et al. [38] adjusted the surprise search algorithm to maintain quality and not only focus on surprise. They utilized the new algorithm (Constrained Surprise Search) to generate a diverse set of balanced weapons for FPS shooter games. They also showcased that it can be used to generate robot controllers, maze solutions, and new mazes [39]. Liapis et al. [61] and Barthet et al. [5] used a denoising autoencoder to help Constrained Novelty search [59] to find innovative

spaceship designs and MineCraft (Mojang Studios, 2009) buildings, respectively. The denoising autoencoder is trained online from the found data and then the compressed representation is used to measure the novelty of the generated content.

25.6.2.2 Offline Methods

On the offline side, Karavalos et al. [51] trained a neural network to predict the win rate and time to finish an FPS match of a given level and the player classes. They trained the model offline on tons of random matches using random levels and random player classes, then later used the trained model as a surrogate model to evolve different player classes. Migkotzidis and Liapis [65] used the same model as part of a mixed-initiative tool [115] that can be used to design balanced levels for FPS games. On the side of affective computing, Barthet et al. [4] use a variant of Go-Explore called Go-Blend (a quality diversity algorithm that keeps track of the best solutions) to explore game trajectories that can mimic different human play styles and arousal levels. They used a simple K-NN algorithm over the AGAIN dataset [64] to predict the arousal levels during playing a car racing game. Similarly, Shaker et al. [88] trained an offline surrogate model on human preference over Mario levels and used it to generate new levels for Super Mario Bros (Nintendo, 1985). Since the search space was not huge, an exhaustive search was used instead of evolution. We included this research as the exhaustive search can be easily replaced with evolution in more complex games.

A different way to use the ML model is to use it to play games instead of directly measuring playability or the attributes and then extracting the needed statistics from the playthrough. For example, Togelius and Schmidhuber [108] evolved arcade game rules using $\mu + \lambda$ evolution strategy to evolve the game rules such that the evolved games are fun to play by humans. Togelius and Schmidhuber use Koster's theory of fun [56] to estimate how learnable these games are. The learnability is approximated by measuring the improvement of a neural network agent that is trained using reinforcement learning. The generated games from that experiment were interesting but not fun for humans. This is due to the fact that AI agents play differently than humans, and we need models that can model the human experience.

Finally, surrogate models can also be used to assist game-playing agents. Olesen et al. [69] tried to use EML to control the player in a car racing experiment. ML was used to learn a forward model which predicts the next state given a certain state and action. This process is called a World Model [41], where we try to learn an approximate forward model of a specific environment. In their work, Olesen et al. used a VAE followed by Mixture Density Network [9] to learn that forward model. Later, they used an evolutionary planning algorithm (Random Mutation Hill Climber (RMHC)) to control the player car to play the game efficiently. The system first gets trained using collected data from random policy, then later it gets improved using frames and output from the RMHC algorithm. Similarly, Dockhorn et al. [23] used hashmaps and decision trees to learn a local forward model for Sokoban. A local forward model is a model that only cares about the local observation around the

player character. They use RHEA to play the game using the local forward model. Although the local forward model has high accuracy, the local model propagates errors over time which causes the planning agent to not achieve very high scores in playing the games.

25.6.3 Machine Learning as an Operator

Models trained via ML can be used as mutation or crossover operators during evolution. This is particularly useful when the space of mutations is very large, making it unlikely that random perturbations will lead to meaningful changes in the genome. In such a scenario, a ML-based model can be trained beforehand so as to learn useful priors over the space of possible mutations. This is particularly appealing when a large dataset of human examples is available.

As an example, ML models can be trained on mutation trajectories [55, 58], i.e. the modifications made to an entity over the course of evolution. The end result is a model that can generate levels by modifying levels similar to an evolutionary/search-based generator.

In another example, the problem of genetic programming is highly complex. A very naive approach could be to try mutating a piece of code at the string or character level, but this would result in a combinatorially explosive action space for the mutation operator. For this reason, most genetic programming approaches will "bake in" some useful priors into the action space afforded to the operator—for example, by providing a higher level set of available actions that may involve adding functional blocks of code such as if/else statements instead of individual characters or words, while treating the code as a graph or tree structure instead of merely a string.

Large Language Models (LLMs) have shown impressive performance on next-token prediction tasks when trained auto-regressively on massive corpora of human text scraped from the Internet. Despite their simple training scheme, they can be observed during inference to generate text that is coherent at a high level, and with the right prompting, can do things like generating rhyming poems about a particular subject matter in a particular style and answering questions while maintaining long-range dependencies. They have also been fine-tuned on corpora of code, and incorporated into tools such as Github Copilot.

In Evolution through Large Models (ELM, Sect. 10.10), the authors take advantage of these learned priors, and prompt LLMs to produce embodied agents capable of traversing the terrain in the SodaWorld environment. The LLM used is a diff model, which is trained on a dataset of code changes and their corresponding natural language descriptions in commit messages. ELM uses this diff model as a mutation operator by randomly selecting from a fixed set of generic commit messages (i.e. "made changes to the code"). This operator is then used inside a quality diversity (QD) loop that searches for a diverse group of ambulating soft robots in SodaWorld. The LLM can

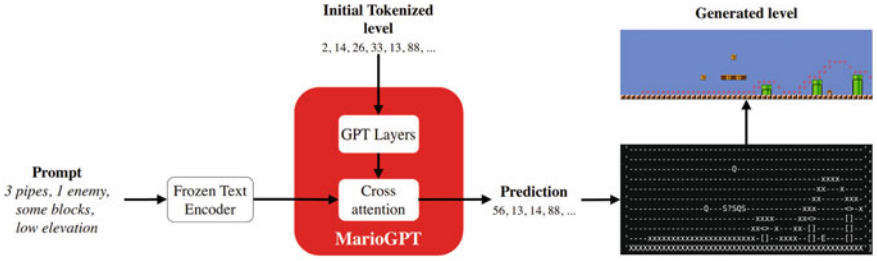


Fig. 25.5 The MarioGPT level-generation pipeline

be additionally fine-tuned on the series of mutations accepted to the archive of elites in QD (an approach similarly applied to environment generation in [55]).

An ELM type approach has also been applied to game levels in MarioGPT [97], where GPT-based language models are fine-tuned on a small dataset of Super Mario Bros (Nintendo, 1985) levels (Fig. 25.5). After fine-tuning the language model, novelty search is used as an outer evolutionary loop to search for a diverse set of novel levels generated by the model. While QD fills an archive spanning over multiple behavior characteristics or measures of interest, novelty search simply looks for sufficiently "different" individuals, with respect to those already in the archive of elites, by computing the mean distance using some distance function in phenotype space. In MarioGPT, the authors consider the mean distance between the paths corresponding to (tree search-generated) solutions for generated levels, where paths are lines over the 2D plane corresponding to the shape of generated levels.

The previous examples are the only ones that we could find specifically on games. It is clear that this area provides ample opportunities for future exploration. For example, one might consider using LLMs for Neural Architecture Search for game-playing agents similar to the work by Chen et al. [15]. Chen et al. [15] represented the neural network architecture as a piece of code that defines layers, skip connections, activations, etc. They used the diff model as a mutation operator to produce new networks. These nets are then evaluated by training them on a labeled dataset (e.g. image classification) and evaluating their test accuracy.

25.7 Augmenting Other Methods with Evolutionary Machine Learning

Finally, EML can be used to support other algorithms like tree search and reinforcement learning. Most of the known work focused on using evolution to find/train neural networks that can support tree search algorithms. Blondie24 [28] is an AI agent that can play checkers very efficiently. It was able to beat 99.61% of the matches that it played against 165 human players. The algorithm uses the Min-Max tree search algorithm [79] with the support of a neural network as a state evaluator. The neural network was trained using an evolutionary algorithm. In a similar manner, Reisinger

et al. [73] not only evolved the network weights but also the architecture using NEAT algorithm [96]. The evolved network was used as a state evaluator for the alpha-beta pruning algorithm [79] to play different board games from the General Game Playing Framework [34]. The best-evolved agent was able to beat the random agent in 5 different games from the General Game Playing framework. Finally, Gauci and Stanely [33] used the HyperNEAT algorithm [95] to prune some branches for the Min-Max algorithm [79] besides the alpha-beta pruning. The final agent was able to have a higher win rate and more ties compared to the default alpha-beta pruning algorithm.

We can notice that most of the work in that area is older than 10 years ago, and we could not find any new work that combines EML with other search algorithms. We think that the boom in computation power and the dependence on the backpropagation algorithm is the main cause of that. For example, the Alpha-Go algorithm [89] uses a neural network as a state evaluator for MCTS agent [11]. This is similar to the Blondie24 agent with the difference of having more computation power to train a big network using backpropagation [44]. This does not mean that we should abandon EML for the sake of backpropagation, but it should push us toward using EML in a smarter and different way that backpropagation cannot do. For example, we could try evolving a network that compresses the game state space such that a dynamic programming agent can play the game efficiently.

25.8 Conclusion

As demonstrated by the myriad examples in this chapter, game research has been enhanced greatly by evolution and ML. The combination of the two approaches provides particularly compelling tools for generating game agents and content; the two types of methods naturally complement each other's strengths and weaknesses. The future of this research area is exciting, as rapid advances in ML technologies will allow us to not only apply new algorithms to existing games but perhaps even to create new kinds of game-based challenges as well (which will, in turn, provide new domains for evaluating new kinds of EML systems).

References

1. Alvernaz, S., Togelius, J.: Autoencoder-augmented neuroevolutionfor visual doom playing. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE, MIT Press (2017)
2. Awiszus, M., Schubert, F., Rosenhahn, B.: Toad-gan: coherent style level generation from a single example. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **16**, 10–16 (2020)
3. Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., Clune, J.: Video pretraining (vpt): learning to act by watching unlabeled online videos. In: Advances in Neural Information Processing Systems (2023)

4. Barthet, M., Khalifa, A., Liapis, A., Yannakakis, G.: Generative personas that behave and experience like humans. In: Proceedings of the 17th International Conference on the Foundations of Digital Games, pp. 1–10 (2022)
5. Barthet, M., Liapis, A., Yannakakis, G.N.: Open-ended evolution for minecraft building generation. In: IEEE Transactions on Games (2022)
6. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: an evaluation platform for general agents. J. Artif. Intell. Res. **47**, 253–279 (2013)
7. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 41–48 (2009)
8. Bhatt, V., Tjanaka, B., Fontaine, M.C., Nikolaidis, S.: Deep surrogate assisted generation of environments (2022). arXiv:2206.04199
9. Bishop, C.M.: Mixture density networks. In: Neural Computing Research Group Report (1994)
10. Bontrager, P., Lin, W., Togelius, J., Risi, S.: Deep interactive evolution. In: Computational Intelligence in Music, Sound, Art and Design: 7th International Conference, EvoMUSART 2018, Parma, Italy, April 4-6, 2018, Proceedings, pp. 267–282. Springer (2018)
11. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–43 (2012)
12. Campbell, M., Joseph Hoane, A., Jr., Hsu, F.: Deep blue. Artif. Intell. **134**(1–2), 57–83 (2002)
13. Cardamone, L., Yannakakis, G.N., Togelius, J., Luca Lanzi, P.: Evolving interesting maps for a first person shooter. In: Applications of Evolutionary Computation: EvoApplications, pp. 63–72. Springer (2011)
14. Carvelli, C., Grbic, D., Risi, S.: Evolving hypernetworks for game-playing agents. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pp. 71–72 (2020)
15. Chen, A., Dohan, D.M., So, D.R.: Evoprompting: language models for code-level neural architecture search (2023). arXiv:2302.14838
16. Chrabaszcz, P., Loshchilov, I., Hutter, F.: Back to basics: benchmarking canonical evolution strategies for playing atari. In: IJCAI (2018)
17. Cook, M., Colton, S., Raad, A., Gow, J.: Mechanic miner: reflection-driven game mechanic discovery and level design. In: Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings 16, pp. 284–293. Springer (2013)
18. Cuccu, G., Togelius, J., Cudré-Mauroux, P.: Playing atari with six neurons (2018). arXiv:1806.01363
19. Dahlskog, S., Togelius, J.: Patterns as objectives for level generation. In: Design Patterns in Games (DPG), Chania, Crete, Greece (2013). ACM Digital Library (2013)
20. Soares de Lima, E., Feijó, B., Furtado, A.L.: Procedural generation of quests for games using genetic algorithms and automated planning. In: SBGames, pp. 144–153 (2019)
21. Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., Levine, S.: Emergent complexity and zero-shot transfer via unsupervised environment design. Adv. Neural Inf. Process. Syst. **33**, 13049–13061 (2020)
22. Dharna, A., Togelius, J., Soros, L.B.: Co-generation of game levels and game-playing agents. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 16, pp. 203–209 (2020)
23. Dockhorn, A., Lucas, S.M., Volz, V., Bravi, I., Gaina, R.D., Perez-Liebana, D.: Learning local forward models on unforgiving games. In: 2019 IEEE Conference on Games (CoG), pp. 1–4. IEEE (2019)
24. Earle, S., Snider, J., Fontaine, M.C., Nikolaidis, S., Togelius, J.: Illuminating diverse neural cellular automata for level generation. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 68–76 (2022)
25. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O., Clune, J.: First return, then explore. Nature **590**(7847), 580–586 (2021)

26. Eigenfeldt, A.: Corpus-based recombinant composition using a genetic algorithm. Soft Comput. **16**, 2049–2056 (2012)
27. Elsken, T., Hendrik Metzen, J., Hutter, F.: Neural architecture search: a survey. J Mach. Learn. Res. **20**(1), 1997–2017 (2019)
28. Fogel, D.B.: Blondie24: Playing at the Edge of AI. Morgan Kaufmann (2002)
29. Fontaine, M.C., Liu, R., Khalifa, A., Modi, J., Togelius, J., Hoover, A.K., Nikolaidis, S: Illuminating mario scenes in the latent space of a generative adversarial network. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 5922–5930 (2021)
30. Fontaine, M.C., Togelius, J., Nikolaidis, S., Hoover, A.K.: Covariance matrix adaptation for the rapid illumination of behavior space. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 94–102 (2020)
31. Gaina, R.D., Lucas, S.M., Perez-Liebana, D.: Rolling horizon evolution enhancements in general video game playing. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG), pp. 88–95. IEEE (2017)
32. Gallotta, R., Arulkumaran, K., Soros, L.B.: Preference-learning emitters for mixed-initiative quality-diversity algorithms. In: IEEE Transactions on Games, pp. 1–14 (2023)
33. Gauci, J., Stanley, K.O.: Evolving neural networks for geometric game-tree pruning. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 379–386 (2011)
34. Genesereth, M., Love, N., Pell, B.: General game playing: overview of the aaai competition. AI Mag. **26**(2), 62–62 (2005)
35. Golberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addion wesley (1989)
36. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Bing, X., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Commun. ACM **63**(11), 139–144 (2020)
37. Gravina, D., Liapis, A., Yannakakis, G.: Surprise search: beyond objectives and novelty. Proc. Genet. Evol. Comput. Conf. **2016**, 677–684 (2016)
38. Gravina, D., Liapis, A., Yannakakis, G.N.: Constrained surprise search for content generation. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2016)
39. Gravina, D., Liapis, A., Yannakakis, G.N.: Quality diversity through surprise. IEEE Trans. Evol. Comput. **23**(4), 603–616 (2018)
40. Guss, W.H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., Salakhutdinov, R.: Minerl: a large-scale dataset of minecraft demonstrations (2019). arXiv:1907.13440
41. Ha, D., Schmidhuber, J.: World models (2018). arXiv:1803.10122
42. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). Evol. Comput. **11**(1), 1–18 (2003)
43. Jonathan Hastings, E., Guha, R.K., Stanley, K.O.: Automatic content generation in the galactic arms race video game. IEEE Trans. Comput. Intell. AI Games **1**(4), 245–263 (2009)
44. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural networks for perception, pp. 65–93. Elsevier (1992)
45. Holmgård, C., Liapis, A., Togelius, J., Yannakakis, G.N.: Evolving personas for player decision modeling. In: 2014 IEEE Conference on Computational Intelligence and Games, pp. 1–8. IEEE (2014)
46. Holmgård, C., Liapis, A., Togelius, J., Yannakakis G.N.: Evolving models of player decision making: personas versus clones. Entertain. Comput. **16**, 95–104 (2016)
47. Hong, S.-W., Kim, J.-T., Kim, H.-I.: Identification of auto programs by using decision tree learning for mmorpg. J. Korea Multimed. Soc. **9**(7), 927–937 (2006)
48. Simon, J.: Artbreeder (2018). https://www.artbreeder.com/. Accessed: March 17, 2023
49. Justesen, N., Bontrager, P., Togelius, J., Risi, S.: Deep learning for video game playing. IEEE Trans. Games **12**(1), 1–20 (2019)

50. Justesen, N., Mahlmann, T., Togelius, J.: Online evolution for multi-action adversarial games. In: Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30–April 1, 2016, Proceedings, Part I 19, pp. 590–603. Springer (2016)
51. Karavolos, D., Liapis, A., Yannakakis, G.N.: Using a surrogate model of gameplay for automated level design. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2018)
52. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaśkowski, W.: Vizdoom: a doom-based ai research platform for visual reinforcement learning. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2016)
53. Khalifa, A., Bontrager, P., Earle, S., Togelius, J.: Pcgrl: procedural content generation via reinforcement learning. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **16**, 95–101 (2020)
54. Khalifa, A., Cerny Green, M., Barros, G., Togelius, J.: Intentional computational level design. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 796–803 (2019)
55. Khalifa, A., Togelius, J., Cerny Green, M.: Mutation models: learning to generate levels by imitating evolution. In: Proceedings of the 17th International Conference on the Foundations of Digital Games, pp. 1–9 (2022)
56. Koster, R.: Theory of Fun for Game Design. O'Reilly Media, Inc. (2013)
57. Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE J **37**(2), 233–243 (1991)
58. Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., Stanley, K.O.: Evolution through large models (2022). arXiv:2206.08896
59. Lehman, J., Stanley, K.O.: Abandoning objectives: evolution through the search for novelty alone. Evolut. Comput. **19**(2), 189–223 (2011)
60. Lehman, J., Stanley, K.O.: Novelty search and the problem with objectives. In: Genetic Programming Theory and Practice IX, pp. 37–56 (2011)
61. Liapis, A., Martínez, H.P., Togelius, J., Yannakakis, G.N.: Transforming exploratory creativity with delenox (2021). arXiv:2103.11715
62. Liapis, A., Yannakakis, G.N., Togelius, J.: Generating map sketches for strategy games. In: Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings 16, pp. 264–273. Springer (2013)
63. Liu, B., Zhang, Q., GE Gielen, G.: A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. IEEE Trans. Evolut. Comput. **18**(2), 180–192 (2013)
64. Melhart, D., Liapis, A., Yannakakis, G.N.: The affect game annotation (again) dataset, pp. arXiv–2104 (2021)
65. Migkotzidis, P., Liapis, A.: Susketch: surrogate models of gameplay as a design assistant. IEEE Trans. Games **14**(2), 273–283 (2021)
66. Minka, T., Cleven, R., Menke, R.: Machine learning for optimal matchmaking (2020). https://www.youtube.com/watch?v=Q8BX0nXfPjY
67. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
68. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT press (2000)
69. VAN Olesen, T., Nguyen, D.T.T., Palm, R.B., Risi, S.: Evolutionary planning in latent space. In: Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24, pp. 522–536. Springer (2021)
70. Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E., Rocktäschel, T.: Evolving curricula with regret-based environment design. In: International Conference on Machine Learning, pp. 17473–17498. PMLR (2022)

71. Perez, D., Samothrakis, S., Lucas, S., ohlfshagen, P.: Rolling horizon evolution versus tree search for navigation in single-player real-time games. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 351–358 (2013)
72. Perez-Liebana, D., Sajid Alam, M., Gaina, R.D.: Rolling horizon neat for general video game playing. In: 2020 IEEE Conference on Games (CoG), pp. 375–382. IEEE (2020)
73. Reisinger, J., Bahceci, E., Karpov, I., Miikkulainen, R.: Coevolving strategies for general game playing. In: 2007 IEEE Symposium on Computational Intelligence and Games, pp. 320–327. IEEE (2007)
74. Jimenez Rezende, D., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. In: International Conference on Machine Learning, pp. 1278–1286. PMLR (2014)
75. Risi, S., Lehman, J., D'Ambrosio, D., Hall, R., Stanley, K.: Combining search-based procedural content generation and social gaming in the petalz video game. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **8**, 63–68 (2012)
76. Risi, S., Togelius, J.: Neuroevolution in games: state of the art and open challenges. IEEE Trans. Comput. Intell. AI Games **9**(1), 25–41 (2015)
77. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models (2021)
78. Rothlauf, F., Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Springer (2006)
79. Russell, S.J.: Artificial Intelligence a Modern Approach. Pearson Education, Inc. (2010)
80. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning (2017). arXiv:1703.03864
81. Samuel, A.: Some studies in machine learning using the game of checkers. Reprinted in Feigenbaum, E.A., Feldman, J. (eds.) (1963). Computers and thought (1959)
82. Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Muller, M., Lake, R., Paul, L., Sutphen, S.: Checkers is solved. Science **317**(5844), 1518–1522 (2007)
83. Schrum, J., Capps, B., Steckel, K., Volz, V., Risi, S.: Hybrid encoding for generating large scale game level patterns with local variations. In: IEEE Transactions on Games (2022)
84. Sfikas, K., Liapis, A.: Playing against the board: rolling horizon evolutionary algorithms against pandemic. IEEE Trans. Games **14**(3), 339–349 (2021)
85. Shaker, N., Nicolau, M., Yannakakis, G.N., Togelius, J., O'neill, M.: Evolving levels for super mario bros using grammatical evolution. In: 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 304–311. IEEE (2012)
86. Shaker, N., Togelius, J., Nelson, M.J.: Procedural Content Generation in Games. Springer (2016)
87. Shaker, N., Togelius, J., Nelson, M.J., Liapis, A., Smith, G., Shaker, N.: Mixed-initiative content creation. In: Procedural Content Generation in Games, pp. 195–214 (2016)
88. Shaker, N., Yannakakis, G., Togelius, J.: Towards automatic personalized content generation for platform games. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **6**, 63–68 (2010)
89. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
90. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
91. Smith, A.M., Lewis, C., Hullet, K., Smith, G., Sullivan, A.: An inclusive view of player modeling. In: Proceedings of the 6th International Conference on Foundations of Digital Games, pp. 301–303 (2011)
92. Sobester, A., Forrester, A., Keane, A.: Engineering Design via Surrogate Modelling: a Practical Guide. Wiley (2008)
93. K.O Stanley. Compositional pattern producing networks: a novel abstraction of development. *Genet. Program. Evolvable Mach.* **8**, 131–162 (2007)

94. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the nero video game. IEEE Trans. Evolut. Comput. **9**(6), 653–668 (2005)
95. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009)
96. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolut. Comput. **10**(2), 99–127 (2002)
97. Sudhakaran, S., González-Duque, M., Glanois, C., Freiberger, M., Najarro, E., Risi, S.: Mariogpt: open-ended text2level generation through large language models (2023). arXiv:2302.05981
98. Summerville, A., Mateas, M.: Mystical tutor: a magic: the gathering design assistant via denoising sequence-to-sequence learning. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **12**, 86–92 (2016)
99. Summerville, A., Mateas, M.: Super mario as a string: platformer level generation via lstms (2016). arXiv:1603.00930
100. Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A.K., Isaksen, A., Nealen, A., Togelius, J.: Procedural content generation via machine learning (pcgml). IEEE Trans. Games **10**(3), 257–270 (2018)
101. Tanabe, T., Fukuchi, K., Sakuma, J., Akimoto, Y.: Level generation for angry birds with sequential vae and latent variable evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1052–1060 (2021)
102. Adaptive Agent Team, Bauer, J., Baumli, K., Baveja, S., Behbahani, F., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., et al.: Human-timescale adaptation in an open-ended task space (2023). arXiv:2301.07608
103. Open Ended Learning Team, Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., et al.: Open-ended learning leads to generally capable agents (2021). arXiv:2107.12808
104. Togelius, J., Karakovskiy, S., Koutník, J., Schmidhuber, J.: Super mario evolution. In: 2009 Ieee Symposium on Computational Intelligence and Games, pp. 156–161. IEEE (2009)
105. Togelius, J., Lucas, S.M.: Evolving controllers for simulated car racing. In: 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1906–1913. IEEE (2005)
106. Togelius, J., Lucas, S.M.: Evolving robust and specialized car racing skills. In: 2006 IEEE International Conference on Evolutionary Computation, pp. 1187–1194. IEEE (2006)
107. Togelius, J., Schaul, T., Wierstra, D., Igel, C., Gomez, F., Schmidhuber, J.: Ontogenetic and phylogenetic reinforcement learning. Künstliche Intelligenz **23**(3), 30–33 (2009)
108. Togelius, J., Schmidhuber, J.: An experiment in automatic game design. In: 2008 IEEE Symposium On Computational Intelligence and Games, pp. 111–118. IEEE (2008)
109. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: a taxonomy and survey. IEEE Trans. Comput. Intell. AI Games **3**(3), 172–186 (2011)
110. Torrey, L., Shavlik, J.: Transfer learning. In: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, pp. 242–264. IGI global (2009)
111. Volz, V., Rudolph, G., Naujoks, B.: Demonstrating the feasibility of automatic game balancing. Proc. Genet. Evolut. Comput. Conf. **2016**, 269–276 (2016)
112. Volz, V., Schrum, J., Liu, J., Lucas, S.M., Smith, A., Risi, S.: Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 221–228 (2018)
113. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: Paired open-ended trailblazer (poet): endlessly generating increasingly complex and diverse learning environments and their solutions (2019). arXiv:1901.01753
114. Wiens, A.L., Ross, B.J.: Gentropy: evolving 2d textures. Comput. Graph. **26**(1), 75–88 (2002)
115. Yannakakis, G.N., Liapis, A., Alexopoulos, C.: Mixed-initiative co-creativity. In: Foundations of Digital Games. ACM (2014)
116. Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer (2018)

117. Zaltron, N., Zurlo, L., Risi, S.: Cg-gan: an interactive evolutionary gan-based approach for facial composite generation. Proc. AAAI Conf. Artif. Intell. **34**, 2544–2551 (2020)
118. Zhang, Y., Fontaine, M.C., Hoover, A.K., Nikolaidis, S.: Deep surrogate assisted map-elites for automated hearthstone deckbuilding. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 158–167 (2022)