applied sciences

MDPI

Review

A Survey of Planning and Learning in Games

Fernando Fradique Duarte [1,*], Nuno Lau [2], Artur Pereira [2] and Luis Paulo Reis [3]

[1] Institute of Electronics and Informatics Engineering of Aveiro (IEETA), University of Aveiro, 3810-193 Aveiro, Portugal
[2] Department of Electronics, Telecommunications and Informatics, University of Aveiro, 3810-193 Aveiro, Portugal; nunolau@ua.pt (N.L.); artur@ua.pt (A.P.)
[3] Faculty of Engineering, Department of Informatics Engineering, University of Porto, 4099-002 Porto, Portugal; lpreis@fe.up.pt
* Correspondence: fjosefradique@ua.pt

Received: 29 May 2020; Accepted: 27 June 2020; Published: 30 June 2020

Abstract: In general, games pose interesting and complex problems for the implementation of intelligent agents and are a popular domain in the study of artificial intelligence. In fact, games have been at the center of some of the most well-known achievements in artificial intelligence. From classical board games such as chess, checkers, backgammon and Go, to video games such as Dota 2 and StarCraft II, artificial intelligence research has devised computer programs that can play at the level of a human master and even at a human world champion level. Planning and learning, two well-known and successful paradigms of artificial intelligence, have greatly contributed to these achievements. Although representing distinct approaches, planning and learning try to solve similar problems and share some similarities. They can even complement each other. This has led to research on methodologies to combine the strengths of both approaches to derive better solutions. This paper presents a survey of the multiple methodologies that have been proposed to integrate planning and learning in the context of games. In order to provide a richer contextualization, the paper also presents learning and planning techniques commonly used in games, both in terms of their theoretical foundations and applications.

Keywords: planning; learning; artificial intelligence; planning and learning; games

1. Introduction

Games have always been widely used as a development and testing environment in the study of artificial intelligence (AI) in academia [1]. The fact that games are well defined by explicit rules, vary greatly in the challenges they pose (e.g., the challenges posed by a puzzle game are very different from those posed by a real time strategy (RTS) game) and that the techniques developed in the domain of games can be transferred to other research fields (e.g., education, psychology) are some of the aspects that have greatly contributed to this [1]. While initial research on game AI focused particularly on classical board games such as chess and checkers, more recently video games have attracted significant research interest, a fact proven by the several international competitions (e.g., the General Video Game AI (GVGAI) competition [2], the General Game Playing (GGP) competition [3], and the Geometry Friends Game AI competition [4]) held annually in multiple international conferences dedicated to game AI (e.g., the Conference on Games (CoG), formerly Conference on Computational Intelligence and Games (CIG), the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), and the International Conference on the Foundations of Digital Games (FDG)). This growing interest has also been fueled by an increasing commercial interest by the gaming industry on finding solutions to create games that are more enjoyable, challenging and engaging, and that ultimately provide a more personalized experience to the human players, in the form of, for example, more sophisticated

AI (i.e., non-player characters or NPCs) that exhibit more complex behaviors and skills and act more naturally and human-like [1,5–8].

In terms of academic accomplishments, games have been at the center of some of the most astonishing and well-known achievements in AI. Ranging from classical board games to video games, AI research has devised computer programs that can play at human master level and even at human world champion level. Examples of these achievements include IBM's Deep Blue [9], the computer program that defeated the world chess champion in 1997; Chinook [10], the World Man-Machine Checkers Champion and the first program to win a human world championship in 1994; Logistello [11], the computer program that defeated the human Othello world champion in 1997; and Quackle, the computer program that defeated the former Scrabble world champion in 2006 [12]. Other examples include KnightCap [13] and Meep [14] in chess and TD-Gammon [15] in backgammon, all of which were able to achieve master level play. More recent examples include the Deep Q-Network (DQN) [16] algorithm, which achieved human level gameplay and above on several games from the Atari 2600 gaming platform, using raw images as input; AlphaGo [17], the computer program that defeated the human European Go champion in 2015; OpenAI Five [18], the first AI to defeat the world champions at an e-sport game, Dota 2, in 2019; and AlphaStar [19], the first AI to defeat a top professional player in StarCraft II in 2018.

Planning and learning, two well-known paradigms and subfields of AI, have contributed greatly to these achievements. Planning is concerned with the implementation of computational methods (or computer programs) that can devise plans (i.e., sequences of actions) in order to achieve a specified goal or a set of goals, whereas learning is focused on the implementation of computer programs that automatically improve their performance over time on future tasks by making observations about the world [12]. While both paradigms have achieved great success in the context of game AI, they both present some shortcomings. Planning techniques, for example, require a model of the environment as well as an accurate description of the planning task (e.g., the specification of the actions that can be performed) to plan, which may not be readily available in more complex environments, such as those featured in video games [20]. In the case of learning techniques, when dealing with more complex environments, such as video games, the learning process may become too slow and fail to derive a solution within a practical time frame, if not provided any guidance (e.g., the agent may spend too much time exploring suboptimal actions) [21]. Both paradigms can however complement each other, in the sense that each paradigm may be leveraged to address the shortcomings of the other [22]. This in turn has led to research on methodologies to combine the two paradigms, leveraging the strengths of both in order to devise better solutions.

The objective of this paper is therefore to present a survey of the multiple methodologies that have been proposed to integrate planning and learning in the context of game AI research. The remainder of the paper is structured as follows. Section 2 begins by introducing some of the most common research topics in game AI. Next, the following two sections are devoted to the presentation of learning (Section 3) and planning (Section 4), respectively. The objective of these two sections is to present some of the most well-known and commonly used techniques from both paradigms, particularly in the context of game AI research. Each of these techniques is briefly presented in terms of its theoretical foundations and applications in games. Following this discussion, Section 5 presents methodologies proposed to integrate both planning and learning techniques. These methodologies are discussed from three main points of view, namely, the integration of planning in order to improve learning and, conversely, the integration of learning in order to improve planning, and, lastly, the combination of both techniques to devise better solutions. Finally, Section 6 presents the conclusions.

While this paper intends to present a review that is as complete as possible, some details will be skipped during the discussion. The interested reader should refer to the cited papers for more information. Furthermore, it should be noted that specific aspects such as memory requirements, algorithmic complexity, learning wall time, and performance rate will not be discussed during the presentation of the techniques reviewed. There are two main reasons for this. First, including such

an analysis would imply choosing an appropriate testing subject (i.e., a specific subset of algorithms) as well as an appropriate testing environment (i.e., a research topic and/or a game genre). This in turn would require careful thought not only in the choice of the most representative algorithms for each technique, given that some of these techniques (e.g., reinforcement learning) are characterized by a thriving (and ever-growing) ecosystem of algorithms, but also to ensure that the techniques are compared on equal terms, since some research topics and game genres, due to their intrinsic characteristics, may favor some techniques over others. This level of complexity by itself (not to mention the relevance and importance of such an analysis) justifies a more appropriate and thorough study in a dedicated paper. Lastly, the objective of the paper is to present a broad overview of the state of the art concerning planning and learning research in games, without being specific to any particular algorithm, research topic, or game genre. The inclusion of this analysis (also because of the reasons already mentioned) could potentially divert the discussion from this objective by focusing too much on specific details associated with these aspects.

2. Research Topics in Game AI

This section presents a brief overview of some topics of research in game AI. This intends to be a cursory introduction to these topics so that the discussion that follows in the remainder of the paper can be better understood.

2.1. Dynamic Difficulty Adjustment

The goal of Dynamic Difficulty Adjustment (DDA) is to try to dynamically adapt the various systems of the game world according to the player's abilities during the course of a gaming session, with the objective of increasing enjoyment. In other words, DDA attempts to adjust the difficulty of the game to match the skill of the player in order to strike a balance between possible boredom (e.g., a highly skilled player playing a game that is too easy) and potential frustration (e.g., a low/medium skilled player playing a game that is too hard) [23]. DDA can be achieved via a plethora of different approaches such as by using models of the player to adjust the AI agents [24], generating levels with the appropriate difficulty via procedural content generation (presented in Section 2.9) [25], and varying the 'intelligence' of the AI agents in order to match the skill of the player [26].

2.2. Game Content Recommendation

Currently, the amount of derived content related to video games can be overwhelming. Players can buy additional downloadable content (DLCs) to personalize or expand their games, exchange game items, stream gameplay in online streaming services (e.g., Twitch), or share their interests in games using social networking services (e.g., Facebook, YouTube). Finding content according to a player's preferences can therefore prove to be a difficult task, given the overwhelming amount of content available. This problem is addressed by Game Content Recommendation (GCR) research whose goal is to build recommendation systems that can automatically recommend game content to players according to their preferences, such as via game replays [27].

2.3. General Video Game Playing

The goal of General Video Game Playing (GVGP) is to create AI agents that can play well in a variety of different games, without having to be specifically programmed or (reprogrammed) to play any of those games (i.e., the agent does not know in advance which games it will play). This is an important area of research in game AI, reflected in several competitions, such as the General Video Game AI competition [2] and the International General Game Playing competition [3].

2.4. Macromanagement and Micromanagement in RTS games

The objective of the player in an RTS game such as StarCraft, a strategic military combat simulation that takes place in a science fiction setting, is to build an army and destroy his enemy's base. This in turn can be highly challenging in terms of the decision-making process as the player must decide which resources to collect (i.e., minerals and gas), which units (e.g., workers or combat units) and buildings (e.g., barracks) to build and in which order, which technologies to pursue, and how to manage the individual units during combat. In this context, macromanagement (MaM) refers to the management of resources and unit production (e.g., build order), whereas micromanagement (MiM) refers to the individual control of units during combat [28].

2.5. Opponent Modeling

Generally speaking, the goal of Opponent Modeling (OM) is to learn a model of the opponent, based for example on his observed actions or behaviors, in order to try to predict his future actions and/or recognize his strategies. This model can be learned offline (e.g., [29]), leveraging the existence of gameplay traces, or online as the game is being played (e.g., [30]). In the context of RTS and fighting games for example, OM can be used in order to derive AI opponents that are more adaptive to the actions and strategies of the player and therefore less predictable, less exploitable, and more interesting and challenging to the human player (e.g., [29–31]).

2.6. Player Modeling

The purpose of Player Modeling (PM) is to learn a model of the player in its many different possible facets (and according to the purpose of the model) such as the player's playstyle [32], playing behavior [33], real world characteristics (e.g., personality traits) [34], playing performance (e.g., game playing expertise), or entertainment preferences [35]. This model can then be used to personalize game content according to the player's personal traits [23], adjust the game's difficulty dynamically [24], improve the player's gaming experience and increase engagement [32], and predict when players will stop playing the game [33], among many other possible applications.

2.7. Learning from Demonstration

The core idea of Learning from Demonstration (LfD), also referred to as Imitation Learning among other designations (e.g., Learning from Observation, Behavioral Cloning), is to use experts (either humans or other artificial agents) to demonstrate to the learning agent how the tasks should be performed. In other words, the agent learns to perform a task by observing an expert performing that same task. LfD can be either passive or active. In passive LfD the agent can only observe the actions of the expert, whereas in active LfD the agent can not only observe the actions of the expert but also query the expert about specific states. Examples of active LfD can be found in [36,37], whereas [38] is an example of passive LfD.

2.8. Transfer Learning

Transfer Learning (TL) is similar to generalization between different tasks. The main idea of TL is to allow the agent to learn new tasks faster by leveraging the previous experience obtained while solving different (but possibly related) tasks [39]. A survey of TL for Reinforcement Learning domains can be found in [40].

2.9. Procedural Content Generation

Procedural Content Generation (PCG) is a research area in game AI concerned with the development of techniques (or algorithms) for the automatic or semi-automatic generation of content for video games (e.g., game levels) [41]. Content creation plays an important role in the video game industry and represents a major portion of the development budget effort of most commercial

games [42]. Most game content is usually generated offline (i.e., during the game development process) but can also be generated online (i.e., after the game has been shipped). Online PCG opens up new and interesting possibilities, such as the generation of content tailored according to the player, which in turn may potentially increase the engagement of the player with the game. As games become increasingly more complex, the use of PCG techniques to help automate content creation can greatly boost the efficiency of the game development process. In fact, many commercial games such as Rogue, Diablo, Torchlight, Spore, Minecraft, and Borderlands have used procedurally generated content [41,42].

2.10. Game Design Support Tools

Game Design Support Tools (GDST) is an umbrella term used here to aggregate work with the objective of implementing tools or techniques that can be used to improve the efficiency (e.g., in terms of cost, complexity, and time) of the game design process (excluding work on PCG). Examples of this include the development of tools to allow the game designer/programmer to create or teach the intended behaviors to their game AI characters [43] and developing techniques to automate the playtesting process [44,45].

2.11. Adaptive Game AI

Over time video games have become increasingly more realistic, particularly in terms of the graphical representation of the virtual world where the game is to take place. However, this sense of emersion and realism can easily be broken if the inhabitant characters, referred to as NPCs, with whom the player can or must interact, do not behave in an appropriate manner (e.g., look dumb, lack common sense, and are predictable). The objective of Adaptive Game AI (AGAI) is therefore to devise techniques to create high-quality game AI [46]. Some examples of characteristics pursued in high-quality game AI include the ability to automatically adapt to the player's strategies and playstyle [46–48], exhibit natural behavior appropriate to act as companions of the player [49,50], be able to demonstrate complex general behaviors [5], act in a more human-like manner [6,7], and be more socially believable [8].

3. Learning in Games

Generally, an agent is said to be learning if it improves its performance over time in future tasks after making observations about the world [12]. Learning encompasses a plethora of different techniques and has found successful application in game AI, especially in academia, where it has played an important role in some of the most well-known milestones in AI research. Machine learning (ML) is the subfield of AI concerned with the development of self-learning systems (e.g., computer programs or algorithms) that can automatically learn and improve from experience (e.g., by capturing the knowledge contained in data) [51]. Most, if not all learning techniques (and in particular those presented in the next subsection) can be categorized as sub-areas of ML. A hands-on introduction to ML (with Python) can be found in [51].

The purpose of this section is to present a brief overview of the multiple applications of learning in games. The section is organized as follows. Section 3.1 is devoted to the presentation of some learning techniques commonly used in games. Next, Section 3.2 presents some overall considerations about the learning techniques presented in the previous section and learning research in general in the context of games. Finally, Section 3.3 discusses learning from the point of view of the video game industry.

3.1. Learning Techniques

3.1.1. Supervised and Unsupervised Learning

The goal of Supervised Learning (SL) is to learn a predictive model that can be used to make predictions about future unseen data; see Figure 1. More formally, given a training dataset composed of N input-output pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where x_i is a feature vector in $\mathbb{R}^d$ and y_i is the desired output or label obtained via an unknown function $y = f(x)$, the objective of SL is to derive a

function h that approximates the true function f. When the labels take discrete values (e.g., {yes, no}) the task is referred to as a classification task, otherwise if the labels take continuous values the task is referred to as regression. Decision Trees and K-Nearest Neighbors (KNNs) are two examples of SL algorithms. Unsupervised Learning (UL), on the other hand, works with unlabeled data (i.e., the dataset consists only of the x_i feature vectors). The goal of UL is to try to extract meaningful information from this data (e.g., patterns). K-Means is an example of an UL algorithm. Other examples of learning algorithms, which can be used to solve both SL and UL tasks, include support vector machines (SVMs) and artificial neural networks (NNs). NNs are discussed in the next subsection.

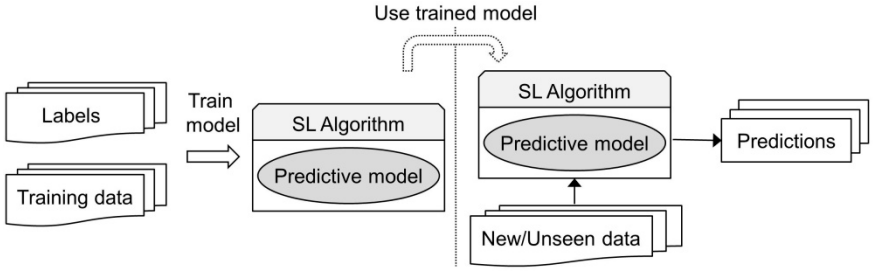

Figure 1. The supervised learning cycle. A labeled dataset (i.e., the training data with the corresponding labels) is used to train a Supervised Learning (SL) algorithm. The resultant predictive model is used to make predictions on new yet unseen data.

SL and UL have found a plethora of applications in the domain of game AI research, ranging from classical board games to video games. A well-known example of this in the domain of classical board games is Logistello [11,52], the computer program that defeated the human Othello world champion in 1997. Logistello uses the Generalized Linear Evaluation Model (GLEM) to automatically find significant features to represent the board positions and derive the evaluation function used to evaluate them (i.e., assign appropriate weights to those features), via SL.

Concerning video games, SL and UL have been used to address game AI-related tasks as diverse as control, player characterization, and content recommendation. Examples of work related to control include General Video Game Playing [53], learning from demonstration in both its active (or online) [37,54] and passive (or offline) [55] variants, and opponent modeling for both offline learning, via the use of gameplay traces in RTS games [29,56] and online learning during actual gameplay in a fighting game [57]. Regarding the characterization of players, these learning techniques have also been used with the objective of predicting aspects of the players as diverse as playstyle [32], playing behavior [33], real world characteristics (e.g., gender and personality traits) [34], playing performance [24], and preferences [27] in order to, for example, adjust the game's difficulty dynamically (e.g., [24]), model player retention (e.g., [58,59]), or implement game content recommendation systems (e.g., [27]).

3.1.2. Neural Networks and Deep Learning

A NN can be thought of as a bio-inspired mathematical model that tries to mimic the way the human brain works [12]. At a high level a NN is composed of an input layer, one or more hidden layers, and an output layer, connected in a sequential way such that the output of a layer serves as the input of the next layer; see Figure 2 (left). In deep learning (DL), a subfield of ML, NNs are usually composed of many hidden layers and are also referred to as deep NNs (DNNs). Each of these layers in turn is composed of a set of artificial neurons. An artificial neuron is a computational unit that tries to mimic the way biological neurons work (e.g., those found in the human brain); see Figure 2 (right). In a feedforward NN, the simplest NN architecture, each artificial neuron in a layer connects to all the artificial neurons in the subsequent layer and there are no connections between artificial neurons within the same layer.

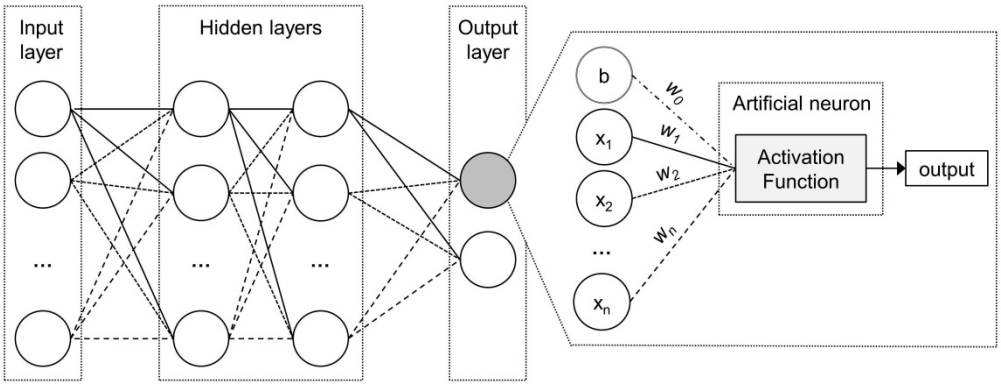

Figure 2. Depiction of a feedforward neural network (NN) with two hidden layers (**left**), showing an artificial neuron in detail (**right**). Each input x_i has an associated weight w_i (the bias term b is associated with weight w_0). The activation value of the neuron is given by $\sum_{i=1}^{n} w_i x_i + w_0 b$ which is then transformed via a nonlinear activation function (e.g., a rectified linear unit or relu) in order to derive the final output value.

NNs can be trained iteratively via a backpropagation algorithm (e.g., stochastic gradient descent or SGD) that tries to optimize a loss (or cost) function (e.g., mean squared error or MSE). The main idea is to compute an error (e.g., the difference between the true or expected values and the output of the network) and use this error as a signal to update the weights w accordingly, throughout the whole network (i.e., the error is propagated backwards through the layers of the network). evolutionary algorithms (EAs), presented in the next subsection, offer yet a different approach to train NNs, often referred to as neuroevolution. Other well-known DNN architectures include convolutional neural networks (CNNs), recurrent neural networks (RNNs), long short-term memory networks (LSTMs) and autoencoders. The interested reader may refer to [60] for an introduction to DL (using Python).

NNs and DNNs have found many applications in the domain of game AI research, both in classical board games and video games. In the domain of classical board games, NNs were used as a way to reduce the need for expert domain knowledge and also to help automate both the process of discovering good features to represent the board positions and the process of deriving the evaluation function used to evaluate those board positions. Examples of this work include game playing programs such as NeuroGammon [61] and [62] in Backgammon, NeuroDraughts [63] in Draughts, and NeuroChess [64] and KnightCap [13] in Chess, some of which managed to achieve master-level or near-expert level play, learning solely via self-play ([62] used SL). In all of the examples given, except [62], the NN was trained via a reinforcement learning (RL) method (i.e., temporal difference (TD)).

Concerning video games, NNs have found applications as diverse as control and learning a model of the environment. Examples of work related to control include macromanagement in RTS games [28] and learning from demonstration, both actively [38] and passively [65]. NNs and, in particular, DNNs have also been used to learn a model of the environment from raw pixel images. An example of this is the work in [66], concerned with spatio-temporal prediction (i.e., predicting image frames given the previous frames and the actions of the agent). More examples are given and discussed later in Section 5.2.4, during the discussion of DNNs applied to planning. Other application uses of NNs include extracting player experience from gameplay videos, transfer learning [67], cheat detection [68], and gameplay recommendation [69].

Finally, the development of new DNN architectures, such as those devised to endow the artificial agent with attention mechanisms [70], imagination [71], curiosity [72], world models that allow the agent to 'hallucinate' inside its own dreams [73], and human-level coordination strategies [74], combined with other learning techniques (e.g., RL and EAs) and planning techniques (e.g., Monte Carlo tree search (MCTS), discussed in Section 4.1.3) may potentially bring AI closer to being able to create artificial agents that behave more like human beings. Some of this work will be further discussed in Section 5 (e.g., [71] and [73] are discussed in Section 5.2.4).

3.1.3. Evolutionary Algorithms

Evolutionary algorithms, a subset of evolutionary computing, are a family of metaheuristic optimization algorithms generally used to solve complex problems (e.g., NP-complete or NP-hard), whose resolution would be too expensive (e.g., in terms of time or computational resources) or even impossible to be obtained otherwise [75]. In general, these solutions do not correspond to the optimal solution but still represent a good approximation to it. EAs are a large family of algorithms, each representing different approaches to solve optimization problems. Genetic algorithms (GAs), a well-known family of EAs, draw inspiration from the processes associated with natural selection (e.g., reproduction, mutation).

The core idea behind GAs is to maintain a set of m candidate solutions (i.e., the population), which is iteratively refined in order to produce better candidate solutions. Each solution in the population is referred to as an individual. At each step of this iterative process an offspring containing k new individuals is derived and added to the population. Each individual in this offspring is obtained by choosing individuals (or parents) from the population, usually in pairs and according to their fitness (i.e., how well they solve the problem), and applying recombination via crossover and mutation in order to derive a new individual combining aspects of both parents. At the end of each step only the m fittest individuals from the total $m + k$ individuals are kept for the next generation (or population) that will be used in the subsequent iteration. This process is repeated k times or until a solution considered good enough is found; see Figure 3.

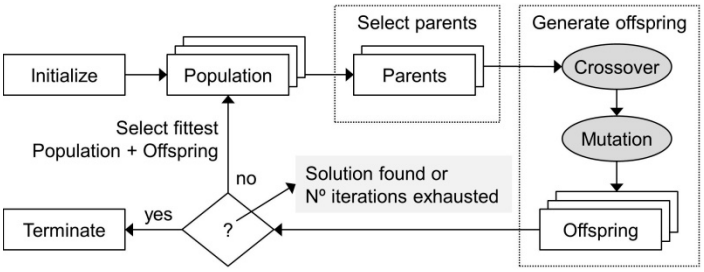

Figure 3. The evolutionary process of a genetic algorithm (GA).

A particular family of EAs, referred to as neuroevolution, is specifically tailored to evolve NNs. Enforced Sub-Populations (ESP), concerned with evolving the weights of the network and Neuroevolution of Augmenting Topologies (NEAT), concerned with evolving not only the weights but also the topology of the network (by complexification), are two such examples. The interested reader may find useful resources on neuroevolution in the context of games in the surveys in references [76,77].

EAs have found diverse applications in games, both in classical board games and video games. In the domain of classical board games, EAs were used, for example, to evolve NN-based agents to play 9×9 Go (e.g., Symbiotic Adaptive Neuro-Evolution (SANE) [78]), chess (e.g., Blondie25 [79]), and checkers (e.g., Anaconda [80]), some of which were able to achieve master (e.g., Blondie25) or even expert level play (e.g., Anaconda). EAs were also used to automatically select appropriate representations for the board positions. An example of this is the checkers-playing agent LS-VisionDraughts [81].

Concerning video games, EAs have also found multiple application uses, including control, player characterization, procedural content generation, game customization, and game design support. Examples of work related to control include imitation learning [82], opponent modeling [31], learning complex general behavior [5], and solving levels of the Jawbreaker puzzle game [83]. Concerning player characterization, EAs were used, for example, to model players in terms of their gameplay experience [84] and entertainment preferences [35]. Regarding the automatic generation of game content, EAs have been used in tasks as diverse as generating the game itself [85,86], game levels [41,87], real-time effects [88], and customized graphical content based on the preferences of the players [89,90].

Automated playtesting is an example of work related to game design support [44,45]. Other applications of EAs include deriving adaptive game AI characters [46,47], cooperative character behavior [49] and human-like behavior [6], evolving multi-agent languages [91], automating the selection of appropriate features in RL problems [92], and enhancing the players' experience in games where the player must control many characters in order to achieve some objective [93].

EAs are also very popular in the GVGAI competition, which also provides an EA-based sample controller. In fact, over the years several EA-based entries have been submitted to the competition with varying degrees of success [2]. Finally, EAs have also inspired the emergence of new game genres. The NeuroEvolving Robotic Operatives game (NERO) [94] is an example of this. In NERO, the objective of the player is to train a team of agents (e.g., an army) through a series of exercises in order to perform a specific task (e.g., fight against an army of agents trained by another human player). This is possible due to the real-time NEAT (rtNEAT) [94] algorithm, whose objective is to evolve increasingly more complex NNs in real-time (i.e., as the game is being played).

3.1.4. Reinforcement Learning

Reinforcement learning is a subfield of ML concerned with the design of agents that learn through trial and error by interacting with their operating environment [95]. Formally, the RL task is formalized as a Markov decision process (MDP) defined as a 4-tuple (S, A, P, R), where S denotes a set of states (i.e., state space), A denotes a set of actions (i.e., action space), $P(s_{t+1}|s_t, a_t)$ represents the state transition probabilities (i.e., the probability of transitioning from state $s_t \in S$ to state $s_{t+1} \in S$ when action $a_t \in A$ is performed at time t), and $R(s,a)$ denotes the reward function (i.e., the immediate reward from taking action $a \in A$ in state $s \in S$). Figure 4 depicts the interaction between the agent and the environment in an MDP.

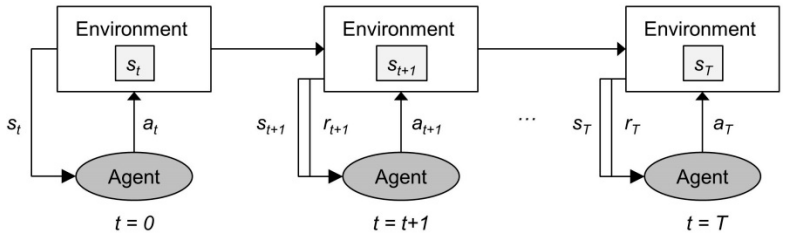

Figure 4. The agent-environment interaction in a Markov decision process (MDP). The agent and the environment interact over a sequence of discrete time steps $t = 0, 1, 2, \ldots$. At each time step t the agent receives a representation of the state of the environment at that time step s_t and performs an action a_t. In response to the action of the agent the environment transitions to the next state s_{t+1}. At the next time step $t + 1$ the agent receives a numerical value (or reward) r_{t+1} assessing its performance and uses this information together with the current state s_{t+1} to choose its next action a_{t+1}.

The objective of RL is to find a policy π (i.e., a mapping between states and actions), defined as $\pi : S \times A \rightarrow [0, 1\}$ if the policy is stochastic or as $\pi : S \rightarrow a \in A(s)$ if the policy is deterministic. The objective of this policy is to maximize the (discounted) return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ over time. $0 \leq \gamma \leq 1$ is the discount rate and $A(s)$ denotes the set of possible actions in state s. In RL, π can be derived via two different approaches: value function and policy gradient. In the value function approach, two different value functions are learned: the state-value function $V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right]$, which computes the expected return when starting in state s and following π thereafter (i.e., the value of the state), and the action-value function $Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\right]$, which computes the expected return when starting in state s, taking action a and following π thereafter (i.e., the value of taking an action in a specific state). $\mathbb{E}$ denotes expectation. These functions can then be used to derive π, e.g., $\pi(s) = \arg max_a Q^{\pi}(s,a)$. In problems with large state spaces, such as games, these functions are usually approximated using NNs, in which case RL is also referred to as deep RL (DRL). Policy gradient methods on the other hand approximate π directly.

Finally, RL methods are catalogued as model-based or model-free, depending respectively on whether they use a model of the environment or not. Q-Learning [96,97] is a well-known example of a model-free RL algorithm. Q-Learning belongs to the family of TD learning (TDL) methods and estimates the Q-values as $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t)]$, where α is the learning rate and $\delta_t = r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t)$ refers to the temporal error. Value Iteration (VI) is a well-known model-based RL algorithm. Model-based RL algorithms can also be thought of as planning algorithms, since they use a model of the environment to simulate actions in order to learn their long-term consequences. Model-based RL is further discussed in Section 5.2.1. Refer to [95] for an in-depth introduction to RL.

Similarly to the learning techniques already discussed, RL has found multiple application uses both in classical board games and video games. Concerning classical board games and additionally to the work already presented in Section 3.1.2 (when presenting applications of NNs in this game genre) more application examples of RL (both in its tabular form and using NNs, i.e., DRL) in this field include TD-net [98] and TD-Gammon [15] in backgammon, Meep [14] in chess, and Samuel's work in checkers [99], some of which achieved master level play (e.g., TD-Gammon, Meep). The reasons behind using RL techniques to tackle this game genre remain the same as before (i.e., reduce the need for expert domain knowledge and help automate both the discovery of good features to represent the board positions and the process of deriving the evaluation functions used to evaluate them). Additionally, RL was also used as a successful alternative to search techniques (e.g., Minimax), in games where these methods suffered from poor performance (e.g., backgammon).

Concerning video games, RL has found multiple and diverse application uses. DRL, in particular, has become quite successful in this domain, since the introduction of DQN [16], a well-known algorithm that managed to achieve human level gameplay and above on several games from the Atari 2600 gaming platform, using raw images as input. Examples of this include work in areas as diverse as deriving adaptive AI agents in RTS games [48], developing natural behaviors for companion NPCs [50], blocking and overtaking in car racing games [100], learning from demonstration [101–103], simultaneous learning (i.e., human feedback is introduced during the RL process) [104,105], simulating crowds [8], procedural content generation [106], and solving the micromanagement task in RTS games [107,108].

Other relevant (and interesting) work includes research on intrinsic motivation [109–111], curiosity via intrinsic rewards generated by the agent itself [112], curriculum learning (i.e., training the agent on increasingly more complex tasks starting from simpler ones) [113], imagination [114], and diverse research in multi-agent environments such as devising coordination strategies in mixed cooperative-competitive environments [115], the use of tools and coordination between agents in competitive settings (with self-supervised auto curriculum) [116], and learning complex skills in simple environments via multi-agent competition [117]. Other recent successes of DRL include OpenAI Five [18], the first AI to defeat the world champions at an e-sport game, Dota 2, in 2019; AlphaStar [19], the first AI to defeat a top professional player in StarCraft II in 2018; and human-level play in the 3D multiagent first person shooter (FPS) video game Quake III Arena in capture-the-flag mode [118].

3.1.5. Other Learning Techniques

Other learning techniques not discussed in the previous subsections include probabilistic models, case-based reasoning (CBR), and dynamic scripting (DS), which are briefly presented next.

Probabilistic Models

Probabilistic models provide a methodology to model and reason (i.e., derive conclusions) about (possibly very complex) systems with incomplete information or uncertainty (e.g., partially observable environments) [119]. A probabilistic model is composed of a set of random variables which model the aspects that characterize the system, and in particular those which are relevant to the reasoning task,

and the corresponding joint distributions representing the interrelationships between these random variables. Probabilistic graphical models (PGMs) provide a methodology to encode probabilistic models compactly in the form of a graph by exploiting the structure of the distribution (i.e., its independence properties). Bayesian networks (BNs), also referred to as belief networks, and Markov networks (MNs) are two examples of PGMs. A comprehensive introduction to graphical models can be found in [119].

Examples of applications of PGMs in the domain of game AI research include opponent modeling in adventure games [120] and RTS games [121–123], and micromanagement in RTS games [124,125]. PGMs have also been used as a game design support tool. An example of this is the work in [43], which proposes a tool to allow game designers to describe, create, tune, and teach (via imitation) different behaviors to NPCs in a first person shooter (FPS) game. These tasks are usually related to imitation learning to some degree or even explicitly (e.g., [43,124]). In general, the parameters of the PGMs are learned via replay data or logs of gameplay traces, usually from human players.

Case-Based Reasoning

Case-based reasoning is one of the five paradigms of machine learning [126]. The core idea of CBR is to represent knowledge in terms of previously experienced situations (i.e., problem situations) or cases and then use these cases to solve new situations. More concretely, the CBR cycle is composed of four processes [127]: retrieve, reuse, revise, and retain. During the retrieve phase, given a new situation a similar case is retrieved from the knowledge-base (or case-base) of previous cases. Next, in the reuse phase the retrieved case is adapted as needed according to the new situation in order to derive a solution. This solution is tested during the revise phase and repaired accordingly in case it fails to solve the situation. The final step is the retain phase where the solutions found to be useful are retained in the knowledge-base for future reuse. An introduction to the foundations of CBR can be found in [127].

CBR has found application in multiple game genres such as classical board games, adventure games and team sports in the context of several different tasks such as puzzle-solving and real-time agent coordination [128]. CBR has also been used in the domain of RTS games. Examples of this include macromanagement [128,129], micromanagement [130], opponent modelling [131], and transfer learning between different game scenarios [39].

Dynamic Scripting

Dynamic scripting [132,133] is an unsupervised online learning technique designed to address both the problems of complexity (i.e., designers must account for all possible in-game situations when designing game AI) and adaptability (i.e., AI should respond accordingly to the changes occurring in the game), most prevalent when using scripts to implement game AI. The core idea in DS is to maintain several rulebases, one for each opponent type in the game, containing manually designed rules. At the start of an encounter, a new script is generated for each opponent, by randomly selecting a specified number of rules from its associated rulebase. The probability of selecting a rule depends on its associated weight. The learning process, inspired by RL, consists of updating the weights of the rules employed, according to their contribution to the final outcome, upon the completion of an encounter (i.e., the weight of each rule is increased or decreased according to whether that rule led to success or failure, respectively).

The practical applicability of DS as an unsupervised online learning tool, more specifically concerning the generation of adaptive opponents at runtime, was assessed in the context of role playing games (RPGs), using both a simulated and a commercial game [132,133]. DS was also used in the context of other game genres, such as RTS games. An example of this is the work in [134], which leverages DS to implement the game AI (the tactics used as rules in the rulebases were evolved offline using an EA).

3.2. Discussion

The previous subsection presented some learning techniques used in the context of game AI and highlighted the different application uses of these techniques in the domain of games. An interesting observation that can be derived from that discussion is the role that learning techniques have played in the context of classical board games, not only as a way of reducing the need for expert domain knowledge and automating some implementation details (e.g., the discovery of features to represent the board positions and the implementation of the evaluation functions used to evaluate them) but also as a successful alternative to search techniques (e.g., Minimax) in games where these techniques suffered from poor performance (e.g., backgammon). In addition, some of these techniques seem to have gained a lot of popularity in solving particular tasks (e.g., EAs in PCG and SL/UL in PM).

Another interesting fact is the prolific integration of DL with both RL and EAs. Since the success of the DQN algorithm, DRL research has gained significant popularity and is currently one of the most popular techniques of choice. In fact, over the years multiple simulation frameworks, such as the Arcade Learning Environment (ALE) [135], VizDoom [136], MuJoCo [137], and Unity's ML-Agents Toolkit [138], have been developed with the purpose of providing challenging environments and benchmarks for DRL research. Furthermore, most research dealing, for example, with endowing AI agents with human-like capabilities such as curiosity, inner motivation, dreaming, and imagination is based on DRL techniques.

In the case of EAs, neuroevolution is a clear proof of the impact caused by DL in the field. Neuroevolution in turn has pushed the boundaries of DL by devising algorithms that can not only train NNs but also build them (e.g., rtNEAT). Neuroevolution has also proven to be a competitive alternative to DRL as attested by Open AI's work in MuJoCo and Atari [139], further enriching the range of possible technical solutions. Table 1 presents an overall comparison of the different learning techniques presented in terms of some of the most relevant aspects that a researcher/practitioner may (or should) take into account when deciding on which technique to use to solve a specific problem related to game AI (or even other kinds of domain).

Table 1. Table presenting a comparison of the different learning techniques discussed. The different aspects comprising the comparison are scored according to an overall score, as these values may differ according to specific algorithms (or variations thereof), research topics, or even game genres. The acronyms used stand for respectively: Domain Expertise Required (DER), Requires a Training Dataset (RTD), Interpretability of the Trained Model (ITM) and Authorial Control (AC). In the table, (1) is used to denote that the need for a training dataset depends on the training method used to train the NNs.

Tech.	DER	RTD	ITM	AC	Popularity
SL/UL	medium/high	yes	low	medium	high
DL	low/medium	(1)	low	low/ medium	very high
EA	low/medium	no	low	low/ medium	very high
RL	low/medium	no	low	low/ medium	very high
PGM	high	yes	medium/ high	medium	medium
CBR	medium/high	yes	medium/ high	medium	low/medium
DS	high	no	high	very high	low/medium

Regardless of this, all of these techniques have found application in the context of games. Table 2 (found at the end of this subsection) presents a brief summary of some of these applications. Section 5 will present even more examples (including some of the most recent milestones in AI history) of the application of these learning techniques in combination with planning techniques.

Table 2. Table presenting some of the most common game AI research topics together with examples of proposed implementations using the learning techniques discussed. To ease the readability of the table the acronyms of the research topics are repeated here. Dynamic Difficulty Adjustment (DDA), Game Content Recommendation (GCR), General Video Game Playing (GVGP), macromanagement (MaM), micromanagement (MiM), Opponent Modeling (OM), Player Modeling (PM), Learning from Demonstration (LfD), Transfer Learning (TL), Procedural Content Generation (PCG), Game Design Support Tools (GDST) and Adaptive Game AI (AGAI).

Research Topics	Learning Techniques
DDA	SL/UL [23,24], DL [140], EA [25]
GCR	SL/UL [27], DL [69]
GVGP	SL/UL [53]
MaM/MiM	PGM [124,125], CBR [129,130], DL [28], RL [107,108]
OM	PGM [121–123], CBR [131], SL/UL [29,56,57], EA [31]
PM	SL/UL [24,32–34,58,59], EA [35,84]
LfD	PGM [43,124], SL/UL [37,54,55], DL [38,65], EA [82], RL [101–105]
TL	CBR [39], DL [67]
PCG	EA [41,85–90], RL [106]
GDST	PGM [43], EA [44,45]
AGAI	EA [5,6,46,47,49], RL [8,48,50], DS [132–134]

Overall, learning has achieved significant success in game AI and has made important contributions to some of the most well-known milestones in AI research, such as Logistello, TD-Gammon, DQN, and more recently OpenAI Five and AlphaStar. As illustrated by these achievements, AI agents are beginning to reach and even surpass human players in some of the most complex video game environments. Learning techniques have also contributed to the enrichment of the gaming ecosystem by fostering new game genres such as NERO. Despite all of these achievements, there is still much work to do in several areas, concerning, for example, the development of techniques to increase data efficiency and significantly speed-up the learning process, particularly in more complex environments such as RTS games, transfer learning between different or related tasks, and devising AI agents that act in a way that is more natural and similar to human players. Further integration with planning techniques may prove to be helpful in dealing with some of these issues (some examples of this are given in Section 5). In addition, there is a need to bridge the gap between the exploits and advancements derived in academia and their adoption by the video game industry, as discussed in the next subsection.

3.3. Learning in Commercial Games

Despite the great success and many achievements of learning in academia, the gaming industry has remained somewhat reluctant to adopt some of these techniques (e.g., DL and DRL), particularly when it comes to modeling agent behaviors. One possible explanation for this may be the lack of expertise of the game designers/programmers concerning some of the techniques developed in academia (e.g., DRL, DL, EAs, PGMs). Budget or time constraints are another possible explanation. In fact, it is estimated that only 20% of all computing resources are available to develop the game AI and, of these, a large portion is spent tuning rudimentary AI behavior, such as maneuvering in the game environment [42].

Yet another explanation may be the tradeoff between authorial control and autonomy. While it can be said that some games and even game genres (e.g., RTS games) thrive on autonomy (e.g., to deliver challenging opponents) it is also true that others purposely minimize autonomy in order to deliver a specific experience to their players (e.g., World of Warcraft) [141]. Furthermore, even when autonomy is desirable, if the AI agent is not properly constrained it can exhibit unpredictable (also referred to as emergent) or inappropriate behavior not anticipated by the game designers/programmers. Such unanticipated behavior in turn may ruin the whole gaming experience. Since AI agents based on NNs, for example, are black boxes for the most part, debugging and preventing such behavior may prove to be a hard task [142]. Of course, these issues do not apply equally to all learning techniques

(e.g., DS generates human-readable scripts that can be manually verified). Overall (and attending to the reasons above), implementing learning in games, whether offline or online, increases the difficulty of implementing and testing the game.

Despite this, learning techniques (NNs in this case) have been used in commercial games such as Black & White from Lionhead Studios (where the objective of the player is to teach and train his own creature) [143] and Supreme Commander 2 (to control the fight or flight behavior of AI controlled platoons) [142] with some degree of success. The use of learning techniques is however not confined to the task of modeling the behaviors of the agents. In fact, these techniques have also been used in the video game industry to help improve the gaming experience of commercial video games via data analytics. Examples of this include the use of ML techniques to cluster players in Battlefield: Bad Company 2 using game telemetry and to create models of the players from gameplay traces both in terms of their performance in World of Warcraft [141] and playing behavior in Tomb Raider: Underworld [33].

4. Planning in Games

Planning is a subfield of AI concerned with the study and implementation of computational methods to devise plans (i.e., find and organize actions) in order to achieve (as best as possible) a specified goal or a set of goals [12]. Another way of defining planning is to think of it as the simulation of possible future interactions with the real environment using an internal model of this environment (or forward model) to learn the long-term consequences of actions [144]. Planning has found successful application in game AI, both in academia and in the video game industry, and has also played an important role in some of the most well-known milestones in AI research

The purpose of this section is to present a brief overview of the many applications of planning in games. The section is organized as follows. Section 4.1 presents some planning techniques commonly used in games. Next, Section 4.2 presents some overall considerations about the planning techniques presented in the previous section and planning research in general in the context of games. Finally, Section 4.3 presents a brief discussion of planning from the point of view of the video game industry.

4.1. Planning Techniques

4.1.1. Search Tree Algorithms

From the perspective of an AI agent, problem-solving can sometimes be simplified by adopting a goal and trying to satisfy it. A goal (i.e., a set of world states in which the goal is satisfied) in turn can be achieved via a sequence of actions (i.e., a solution) leading from the initial state to the goal state. The process of deriving this sequence of actions is referred to as search and can be solved algorithmically (e.g., via an algorithm that takes as input the problem and outputs the solution) [12]. Once a solution has been found it can be used to guide the actions of the agent in order to solve the problem. Search algorithms work by considering several possible action sequences (i.e., solutions), starting from an initial state. This process can be seen as effectively constructing a search tree, rooted at the initial state, whose branches correspond to actions and whose nodes represent the states in the state space of the problem.

More specifically, the search process is carried out as follows. Starting from the root (initial state), the search algorithm tests the current node (or state) against the goal and stops if the goal has been reached. Otherwise, the algorithm expands the current state by performing each of the legal actions in that state and adding the resulting states as its child nodes. The set of (leaf) nodes available for expansion is referred to as the frontier or open list. The search algorithm continues to expand nodes on the frontier until a solution has been found or until there are no more nodes to expand; see Figure 5a. Some examples of this family of algorithms are briefly presented next (more details can be found in [12]).

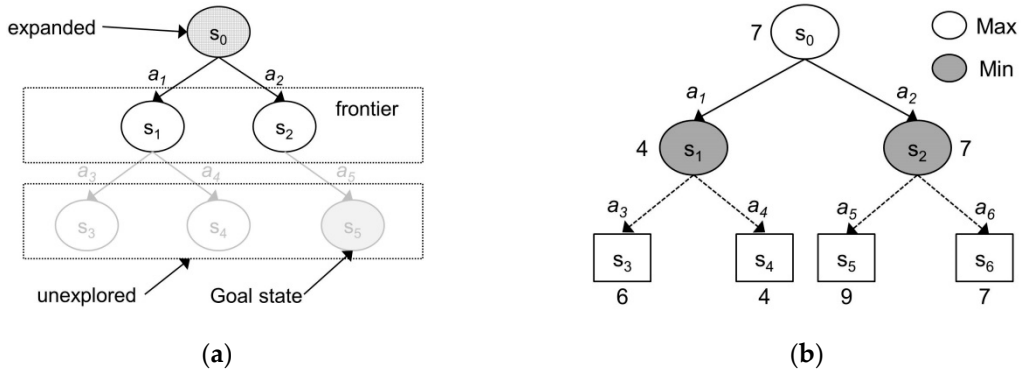

Figure 5. Two examples of search trees: (**a**) A search tree depicting the expansion of the initial state s_0 into two new (child) states s_1 and s_2 (the frontier) derived by performing actions a_1 and a_2, respectively; (**b**) A minimax tree. White nodes are max nodes (max's turn is depicted with a solid line) whereas grey nodes are min nodes (min's turn is depicted with a dashed line). Terminal nodes (squares) show the utility values for max. The remaining nodes show their minimax values. Max's best move at the root is a_2 (highest minimax value of 7), given that it is assumed that min plays optimally (i.e., tries to minimize max's score) and will therefore choose to perform action a_6 (lowest minimax value of 7). The branches of the tree can be thought of as corresponding to the strategies that both max (solid lines) and min (dashed lines) can choose from.

Minimax and Alpha-Beta

In the case of multi-agent competitive environments, such as games, where the goals of the agents conflict with each other, a new kind of search problem arises, referred to as adversarial search. Minimax and Alpha-Beta search are two well-known examples of this family of search methodologies. At a very high level, Minimax considers games where two players, called max and min, respectively, play optimally (taking turns). Max prefers to move to states of maximum value, whereas min prefers states with minimum value. This process builds a minimax tree; see Figure 5b. However, as the number of possible moves each player can choose from (i.e., the branching factor) increases, this tree can grow exponentially, hindering the search process. One way to avoid this problem is to still compute this minimax tree without considering every node in the game tree. This is the core idea behind Alpha-Beta search. In Alpha-Beta search, branches that cannot influence the final decision are pruned from the tree.

Minimax and Alpha-Beta found widespread application in the context of classical board games. Some examples of this were already mentioned in Sections 3.1.2 and 3.1.4 and include, for example, KnightCap [13] and Meep [14] in chess, Logistello [11] in Othello, and Samuel's work in checkers [99]. These techniques also contributed to some of the most well-known milestones in game AI research, such as IBM's Deep Blue [9] in chess and Chinook [10] in checkers. However, the good results obtained by these approaches relied heavily on the use of human expert domain knowledge to devise good heuristics to prune the search space and hand-craft the features to represent the board positions and the evaluation functions used to evaluate them [145,146]. As game AI research moved on to tackle increasingly more complex games such as backgammon, Go, and video games (e.g., RTS games), lacking good known heuristics or featuring larger branching factors, uncertainty, or partial observability, for example, these techniques lost much of their relevance and were replaced by other algorithms such as MCTS (presented in Section 4.1.3) in Go [146] and RL in backgammon [15], more suited to deal with these challenges. Nevertheless, some authors have explored the use of these techniques in some of these more complex scenarios. An example of this is the work in [147], where the authors propose the Alpha-Beta Considering Durations (ABCD) algorithm, an Alpha-Beta search variant devised to deal with RTS combat scenarios.

A*

A* is a well-known best-first search algorithm that incorporates a heuristic function $h(n)$ in its evaluation function $f(n) = g(n) + h(n)$, used to evaluate nodes (or states). In this context $g(n)$ denotes the cost to reach a node, whereas $h(n)$ is used to approximate the cost of reaching the goal from the specified node. A* and its many variants have found successful application in the domain of video games. Examples of this include cooperative pathfinding for real-time environments (e.g., RTS games) [148], realistic path finding that accounts for the different sizes and terrain traversal capabilities of the units [149], and deriving NPC characters with adaptive and more human-like behavior and strategic planning capabilities in RTS games [150]. A* has also been used as a game design support tool to train and author NPC behaviors for open-world video games via player or game designer demonstration [151] and to help verify and tune levels in platformer games) [152]. Finally, A* has also been used for control. An example of this is the successful A*-based controller that was used to play a version of the Super Mario Bros video game [153].

4.1.2. Hierarchical Task Networks

Hierarchical task network (HTN) planning decomposes a planning problem into a set of tasks (e.g., activities or processes) [154]. These tasks can be either primitive, in which case they correspond to an action that may be performed by the planning operators, or non-primitive, in which case they must be decomposed into a set of smaller subtasks, using methods. During the planning procedure, all non-primitive tasks are decomposed recursively into increasingly smaller subtasks until primitive tasks are reached.

More formally, a task network is defined as a pair $w = (U, C)$, where U is a set of task nodes and C is a set of constraints. Each constraint in C specifies a requirement that must be satisfied by every plan that is a solution to the planning problem. In this context, a task is defined as $t(r_1, \ldots, r_k)$, where t is a task symbol and $r_1, \ldots, r_k$ are terms. The task is said to be primitive if t is an operator symbol and non-primitive otherwise. A method is defined as a 4-tuple $m = (name(m), task(m), subtasks(m), constr(m))$, where $name(m)$ is an expression of the form $n(x_1, \ldots, x_k)$, $task(m)$ is a non-primitive task and the 2-tuple $(subtasks(m), constr(m))$ is a task network. In the expression $n(x_1, \ldots, x_k)$, n is a unique method symbol and $x_1, \ldots, x_k$ are all the variable symbols that occur anywhere in m. A graphical depiction of an HTN method is presented in Figure 6.

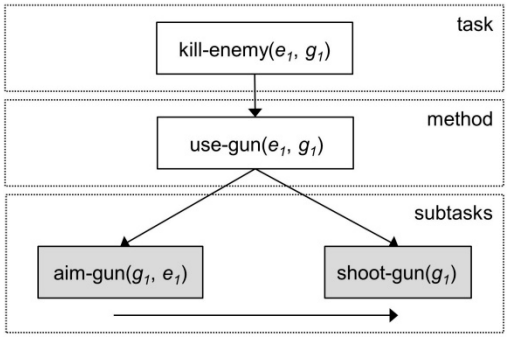

Figure 6. Depiction of the decomposition of task *kill-enemy* (i.e., kill the enemy e_1 using gun g_1) into the subtasks *aim-gun*(g_1, e_1) and *shoot-gun*(g_1) via the hierarchical task network (HTN) method *use-gun*. In this case the agent must first aim the gun before shooting it at the enemy. The order of precedence is depicted by the rightward-pointing arrow. This decomposition forms an AND branch of the decomposition tree. A decomposition tree is a tree structure that represents derivations of plans and results from the combination of these AND-branches.

HTNs have found diverse applications in game AI. One such example concerns the problem of deriving agents that act in a more natural way. Examples of this include creating believable NPCs in serious games [7], generating characters with dynamic behavior in real-time video games [155],

and deriving agents that can act has companions of the human player in Minecraft [156]. Other application uses include tasks as diverse as strategic coordination of teams of agents [157,158], endowing agents with planning capabilities in fighting games [159], computing plans offline that can be used as game scripts [160], interactive storytelling using a model of personality traits [161], and macromanagement in RTS games [162]. HTN planners have also been used in commercial games such as Killzone 3 and Transformers 3: Fall of Cybertron [155].

4.1.3. Monte Carlo Tree Search

Monte Carlo tree search [146] combines tree search with Monte Carlo (MC) evaluation and is an example of decision-time planning. The core idea in MCTS is to iteratively grow a tree that accumulates value estimates obtained from MC simulations. These estimates are used to direct subsequent simulations towards more promising trajectories, making the tree effectively asymmetric. The basic implementation of MCTS is composed of four steps [95]: selection, expansion, simulation, and backup. Starting from the current node (or state), referred to as the root node, and during the selection step, the tree policy traverses the tree until a leaf node is reached. In the expansion step, one or more child nodes are added to this leaf node, further expanding the three. These child nodes consist of states that can be reached from the leaf node via yet unexplored actions. Next, in the simulation step, a complete episode is simulated, starting from the selected node or one of its newly-added child nodes. A rollout policy is used to select the actions during this simulation. Finally, during the backup step, the return generated by the simulated episode is backed up to update the statistics of the nodes traversed by the tree policy during the current iteration; see Figure 7.

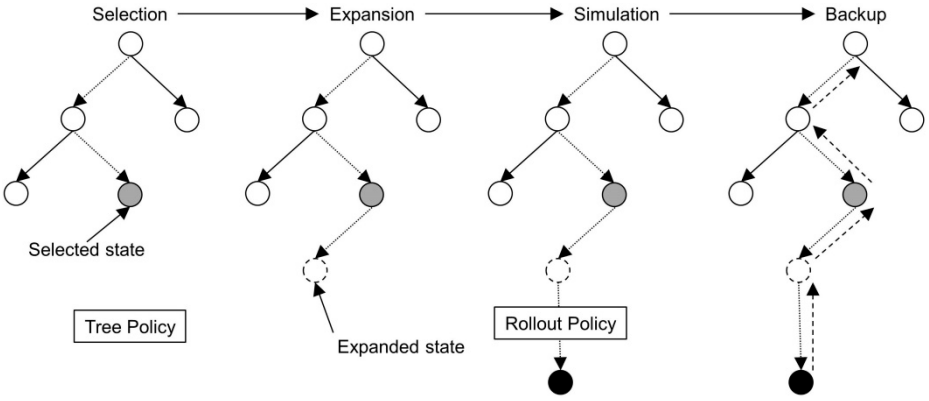

Figure 7. The four steps of the Monte Carlo tree search (MCTS) process: selection, expansion, simulation, and backup.

These four steps continue to be executed (always starting from the root node) until no more time is left or some other computational resource is exhausted, at which time MCTS chooses an action according to some method that depends on the statistics accumulated in the root node (e.g., the action having the highest action value or the action with the largest visit count). Several methods can be used to implement both the tree and rollout policies. A popular implementation of the tree policy is the selection rule used by the upper confidence bounds (UCB) applied to trees (UCT) [163] algorithm (a variant of MCTS), defined as:

$$a^* = \underset{a \in A(s)}{\operatorname{argmax}} \left\{ Q(s,a) + C\sqrt{\frac{lnN(s)}{N(s,a)}} \right\}, \tag{1}$$

where, for a given action a and state s, the statistics accumulated during the simulations are used to derive respectively: $Q(s,a)$, the average reward obtained for a given action a; $N(s,a)$, the number of times action a has been performed in state s; and $N(s)$, the number of times state s has been visited. C is a

constant (also called the exploration constant) used to balance exploration and exploitation. a^* denotes the action that is selected. The rollout policy is commonly implemented via a uniform random selection policy. Over time many different implementations have been proposed for these policies, as well as multiple variations and enhancements to the MCTS algorithm. A survey can be found in [164]. Section 5.2.5 also presents some of this work, in particular that concerning the combination of MCTS with several learning techniques in the context of games.

In the domain of game AI research MCTS was first applied to the game of Go where it achieved very good results (i.e., Crazy Stone [146] in 9×9 Go). MCTS and its variants (e.g., UCT) eventually led to master level play in Go (e.g., MoGo [145] in 9×9 Go) and currently are used as the base of most of the strongest Go programs [165]. These early successes contributed to the popularity of MCTS and its subsequent use to solve a plethora of different challenges in many other game genres. Examples of this include ad-hoc teamwork coordination [166], solving cooperative problems with no prior knowledge [167], dynamically adjusting the difficulty of games [26,168], generating agents that exhibit a more human-like playing style [169,170], micromanagement in RTS games [162,171–173], generating personas for NPCs in fighting games [174], opponent modeling [175], and controlling several agents in constrained learning scenarios applied to serious games [176].

In addition, MCTS has also been used as a sample controller in several competitions such as the Fighting Game AI competition and the General Video Game AI competition. In fact, MCTS has been quite successful in the latter: the winners of the 2016 and 2017 GVGAI Single Player Planning Championship were based on MCTS [177]. More examples of the use of MCTS in GVGP include the work in [178] and [179].

4.1.4. Classical Planning

Planning problems are usually described via a conceptual model and in particular by the model of state-transition systems, defined as a 4-tuple $\Sigma = (S,A,E,\gamma)$ [154]. In the previous definition, S is the set of states, A is the set of actions (i.e., transitions controlled by the plan executor), E is the set of events (i.e., transitions not controlled by the plan executor), and γ: $S \times A \times E \rightarrow 2^S$ is a state-transition function. The planning problem may be simplified by restricting this model via a set of restrictive assumptions such as by assuming that S is finite, Σ is fully observable, actions are instantaneous, or that the planner is not concerned with any changes that might occur in Σ while it is planning.

A family of planning techniques referred to as classical planning assumes a set of eight such assumptions and considers a restricted model of a fully observable, finite, deterministic, and static system with restricted goals and implicit time (i.e., $\Sigma = (S,A,\gamma)$). Another such family is neoclassical planning. Neoclassical planning still deals with restricted models but has introduced new techniques to the planning domain, such as planning-graph techniques, which introduce the planning-graph as a reachability structure in the state space, propositional satisfiability techniques, which treat planning as a satisfiability problem, and constraint satisfaction techniques, which view planning as a constraint satisfaction problem [154]. The term Automated Planning (AP) will be used henceforth to designate both classical and neoclassical planning. Finally, several components of the planning problem (e.g., the actions and the model of the world) are usually expressed via a planning representation language such as the Stanford Research Institute Problem Solver (STRIPS) [180], the Action Description Language (ADL) [181], and the Planning Domain Definition Language (PDDL) [182]; see Figure 8.

Planning techniques dealing with restricted models, such as those discussed previously, are not well suited to cope with challenges such as uncertainty, partial observability, actions with different timespans, or several agents acting simultaneously, common in most video game environments. In fact, their restrictive assumptions exclude these environments. Furthermore, planning representation languages such as PDDL may not be expressive enough. For example, PDDL does not account for the creation or deletion of objects, such as units or buildings in RTS games. This has led to the proposal of several extensions to relax the restrictions imposed on the conceptual model (e.g., the Goal Oriented Action Planner, or GOAP [183], and Layered GOAP, or LGOAP [150]) and to the integration with

techniques borrowed from other AI research fields, such as learning techniques (discussed in Section 5). In addition, several extensions have been proposed to augment PDDL, concerning, for example, the creation and deletion of objects and the use of concurrent actions [184]. Furthermore, new planning languages specific to games have also emerged. An example of this is ABL (A Behavior Language) [185], a reactive planning language designed with the objective of authoring believable agents.

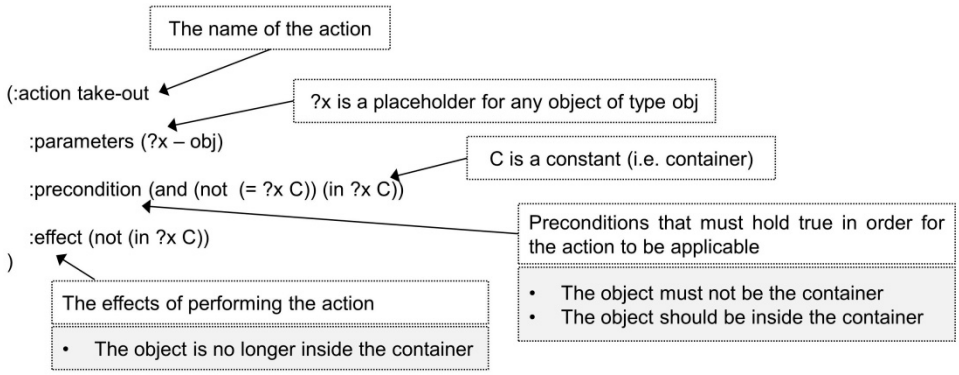

Figure 8. Schema of the action *take-out* defined in Planning Domain Definition Language (PDDL). The objective of this function is to take an object out of a container.

These progresses in turn have allowed AP techniques to find several applications in game AI. Examples of this include deriving agents that can reason about several goals simultaneously (e.g., macro and micromanagement in RTS games) [186], macromanagement in RTS games [187], implementing believable agents [188], deriving more adaptive and challenging AI opponents [189,190], control and coordination of NPCs [191], real-time generation of narratives in interactive storytelling systems [192,193] and [194] (also supporting intentionality and conflict), and adapting human-authored game plotlines offline for role-playing games according to the player's entertainment preferences [195].

4.1.5. Other Planning Techniques

Other planning techniques not discussed in the previous subsections include rapidly-exploring random trees (RRTs), case-based planning (CBP), and behavior trees (BTs), briefly presented next.

Rapidly-Exploring Random Trees

Rapidly-exploring random trees [196] are an efficient data structure and sampling scheme designed to enable quick search in high-dimensional spaces. The key idea is to bias the exploration of the search space towards large unexplored regions by randomly sampling points in space to incrementally construct a search tree that is biased to grow towards these yet unexplored areas. This enables RRTs to rapidly explore and uniformly cover the search space. RRTs have found many applications in game AI. Examples of this include implementing more realistic path planning in RTS games (i.e., the motion constraints of the agents, e.g., vehicles are taken into account) [197], solving NPC companion decision problems in the RPG game Pokémon [198], and solving individual levels of the physics-based platform game Geometry Friends (GF) [199]. RRTs have also been used as a game design support tool. Examples of work in this direction include automating level design [200,201], visualizing and exploring stealth paths in 2D game levels [202], and simulating player behavior in order to explore, test, and tune game levels in platformer games [152]. Finally, RRTs were also used as a tool to evaluate the expressiveness and solution planning capabilities of the action game modeling language HyPED [203].

Case-Based Planning

Case-based planning merges ideas from classical planning with ideas from CBR (already discussed in Section 3.1.5). Similarly to CBR, the core idea of CBP is to reuse past experience (or cases), in this

case encoded in the form of plans, in order to solve new (planning) problems. CBP can therefore be seen as a combination of planning and learning [204,205]. However, in the context of games the use of plans may complicate the CBP cycle, particularly in highly dynamic environments such as RTS games, where generating the solution to the problem (i.e., deriving a plan via the CBP cycle) cannot be decoupled from solving the problem (i.e., executing the plan) and plans may fail during execution due to, for example, inaccuracies in the model of the environment. The On-Line Case-Based Planning (OLCBP) framework [205] proposes a solution to this problem by augmenting the CBP cycle (the OLCBP cycle) in order to incorporate the monitorization of the execution of plans.

Some examples of the use of CBP in the context of game AI include research work aiming to obtain character diversity in terms of behavior (based on plan diversity) [206] and work concerned with endowing the artificial agent with the capability of providing meaningful explanations to its actions [207]. Yet more examples include learning from demonstration [208,209] and opponent modeling in the context of RTS games [210].

Behavior Trees

Behavior trees were developed in the video game industry as an alternative to finite state machines (FSMs) in order to increase modularity in the control structures of NPCs. BTs provide a way to structure task execution (i.e., switching between different tasks) based on conditions and observations and allow the creation of complex systems that are both modular and reactive [211]. Similar to the techniques already discussed, BTs have found several applications in game AI. One of these applications concerns supporting game design. Examples of this include work focused on allowing developers to modify existing AI behaviors dynamically and adapt the driver's getaway AI to the players' skills in the game Driver San Francisco [212], and work on emotional BTs, an extension to BTs that incorporates emotions into the decision-making process of the agents [213]. BTs have also been used to implement dynamic difficulty adjustment in FPS games via the development of adaptive companions that analyze the player's experience and behave accordingly [214]. Finally, several methodologies have been proposed to automatically design BTs using different learning techniques, such as EAs [215,216] and RL [217].

4.2. Discussion

The previous subsection presented some planning techniques used in the context of game AI. The discussion also highlighted the different degrees of adoption and success of these techniques in the domain of games. An interesting point to make in this regard is that while some of the techniques mentioned were tailored specifically for games (e.g., Minimax and MCTS), others were borrowed from other research fields (e.g., CBP, HTNs, and Automated Planning) and consequently had to be extended to cope with the specificities inherit to the virtual environments where games take place, such as uncertainty and partial observability. AP techniques are among those that suffered the most in terms of adoption because of this, although several extensions have been proposed in order to address these difficulties.

Another interesting observation is that while some of these techniques were specifically tailored for games, their different underlying approaches have had a significant impact on their applicability. Minimax and Alpha-Beta are such examples. While having enjoyed large success in classical board games, as fundamental pieces of some of the most well-known milestones in AI (e.g., IBM's Deep Blue and Chinook), when faced with game environments featuring large branching factors or lacking good heuristics such as in Go, uncertainty such as in backgammon, and continuous and/or real time decision making such as in RTS games, some of their underlying limitations ended up being exposed and ultimately proved to be an obstacle to their adoption in more complex games and different game genres. MCTS on the other hand, while initially applied to the game of Go, where it achieved great success, proved to be more extensible and has gained significant popularity and applications in other game genres, such as RTS games. A final interesting remark concerns the use of planning techniques (i.e., HTNs and MCTS) in the domain of virtual reality and, more specifically, serious

games. It should be expectable to see more examples of the integration of planning in the domain of virtual, augmented, and mixed reality in the future. Similarly to what was done regarding learning techniques, Table 3 presents an overall comparison concerning the different planning techniques presented. Again, the aspects used for the comparison represent some of the relevant questions that researchers/practitioners may ask themselves when deciding on which technique to use to solve a specific problem related to game AI.

Table 3. Table presenting a comparison between the different planning techniques discussed. Similarly to what was done previously, while presenting a similar comparison for learning techniques and due to the same reasons, the aspects comprising the comparison are scored according to an overall score. The acronyms used stand for respectively: Requires a Planning Technique (RPL), Complexity of the Forward Model (CFM), Is Anytime Planning (ANTP), Integration with Learning (IWL), and Domain Expertise Required (DER). In the table '—' is used to denote that the aspect is not applicable to that technique.

Tech.	RPL	CFM	ANTP	IWL	DER
Search Tree	no	low/medium	no	easy	medium/high
HTN	yes	high	no	hard	high
MCTS	no	low/medium	yes	easy	low/medium
AP	yes	high	no	hard	high
RRT	no	low/medium	no	easy	low/medium
CBP	no	–	no	easy/medium	medium/high
BT	no	–	–	easy	high

Regardless of this, all of these techniques have found application in the context of games. Table 4, found at the end of this subsection, presents a brief summary of some of these applications. Section 5 presents even more examples of application of these techniques in combination with learning techniques in the context of games.

Table 4. Table presenting some of the most common game AI research topics together with examples of proposed implementations using the planning techniques discussed. The acronyms used in the table stand for respectively: Dynamic Difficulty Adjustment (DDA), General Video Game Playing (GVGP), macromanagement (MaM), micromanagement (MiM), Opponent Modeling (OM), Learning from Demonstration (LfD), Procedural Content Generation (PCG), Game Design Support Tools (GDST), Adaptive Game AI (AGAI), Path Finding (PF), Multi-Agent Coordination (MAC), and Narrative Planning (NP) regarding the research topics and Reactive Planning (RP), Deliberative Planning (DP), Partial Satisfaction Planning (PSP), Epistemic Planning (EP), Temporal Planning (TP), Intentional Narrative Planning (INP), and Partial-Order Planning (POP) for the planning techniques (the acronyms not yet specified). In the table, A* represents both A* and its variants.

Research Topics	Planning Techniques
DDA	MCTS [26,168], BT [214]
GVGP	MCTS [177–179]
MaM/MiM	(HTN, MCTS) [162], MCTS [171–173], RP [186], POP [187]
OM	CBP [210], MCTS [175]
LfD	A* [151], CBP [208,209]
PCG	POP [195]
GDST	(A*, RRT, MCTS) [152], RRT [200–203], BT [212,213]
AGAI	A* [150], RRT [198], CBP [206]. HTN [7,155,156], MCTS [169,170,174,176], (DP, RP) [189], EP [188]
PF	A* [148,149], RRT [197]
MAC	HTN [157,158], MCTS [166,167], PSP [191]
NP	HTN [161], TP [192], EP [193], INP [194]

Overall, and as a conclusion, it can be said that planning has achieved significant success in game AI. Proof of this is the crucial role that planning has performed in achieving several milestones in AI research. Planning has also helped further enrich the gaming ecosystem. Examples of this include the introduction of a new game genre, anticipation games, where planning is used as the key concept both at design time and run time [218], games that rely heavily on planning [219], and prototyping gaming platforms aimed specifically to ease the modeling process of planning agents (social interaction in this case) [220].

Despite this however, there is still plenty left to do, particularly concerning the integration of AP techniques in the context of games. Apart from the difficulties arising from the fact that these techniques use a restricted model of the problem, other difficulties still need to be properly addressed. One such difficulty concerns the different representations used by both domains (mostly declarative in the case of planning techniques and procedural in the case of virtual environments), although some efforts have been made in order to implement tools to bridge the gap between both domains (e.g., Bowyer [221]). Finally, as yet another example, more research efforts must be put into devising systems that not only plan but also act in the sense that they are adaptable and react to changes, detecting plan execution failures and/or unsuitability, due to the changes occurring in the environment, and replanning accordingly [222]. The Planning Execution and LEarning Architecture (PELEA) [223,224] is an example of work that addresses these challenges.

Ultimately, integrating automated planning techniques in the domain of games can prove to be highly beneficial since these techniques deal with higher abstractions to represent the world, such as objects and relationships between objects, which in turn allow more powerful reasoning techniques such as propositional logic and first-order-logic to be leveraged during the reasoning process. The Planning Execution Observation Reinforcement Learning (PEORL) [225] framework is an example of work in this direction.

4.3. Planning in Commercial Games

F.E.A.R. is considered to be one of the first commercial video games to successfully use a planner (GOAP [183]) to model the behavior of agents, at a time where the most commonly used techniques to define the behaviors of NPCs were FSMs and BTs [226]. While these techniques give full authoring control to the designer and allow agents to be reactive, they also have some drawbacks, for example the agents cannot plan further ahead in time (i.e., long-term plans), and support for multi-agent coordination is also very limited. Planners were therefore introduced in video games in order to address some of these shortcomings.

After the success of F.E.A.R., many other games started to implement action planners. Examples of such commercial games in which planners were used to model the behavior of the NPCs and, in some cases, also to coordinate them, include titles such as [226] Dirty Harry, Tomb Raider, Middle-Earth: Shadow of Mordor, Killzone 2 (the first to use HTNs), Killzone 3, Transformers: Fall of Cybertron, Dying Light, and PlannedAssault, a mission generator for ARMA II. These planners, as well as most of the planners used in commercial video games, use some modified and or extended version of STRIPS or HTNs [226]. Other examples of planning techniques used in commercial video games include A* for path finding, a task in which it has achieved great popularity [183], and MCTS to create interesting and strong AI players that can play more like a human player in the AI Factory's Mobile Card game Spades [170].

5. Planning and Learning in Games

As outlined in the previous sections, planning and learning techniques have found a wide variety of successful applications in the domain of games. However, both paradigms present some shortcomings. Planning techniques for example, require a model of the environment as well as an accurate description of the planning task (e.g., the specification of the actions that can be performed) to plan, which may not be readily available in more complex environments, such as those featured in

video games [20]. In the case of learning techniques, when dealing with more complex environments, such as video games, the learning process may become too slow and fail to derive a solution within a practical time frame, if not provided any guidance (e.g., the agent may spend too much time exploring suboptimal actions) [21]. Both paradigms may however be combined to address these shortcomings. Although representing distinct research areas, planning and learning try to solve similar problems and share some similarities in terms of techniques and methodologies. In fact, they can complement each other [22]. As an example of this, and in the case of the shortcomings given as examples earlier on, learning techniques can be used to learn a world model, whereas planning techniques can be used to guide the exploration of the learning process towards more promising areas of the state space.

This has led many researchers to explore different methodologies to combine the techniques from both paradigms, leveraging their individual strengths in order to derive better solutions. This section presents some of this work in the context of game AI under three different perspectives: Section 5.1 discusses the use of planning techniques to improve the learning process; conversely, Section 5.2 discusses the use of learning techniques to improve and enhance planning techniques; and, finally, Section 5.3 discusses the combination of both paradigms in order to derive better solutions. Section 5.4 concludes the discussion and presents some overall considerations and final remarks.

5.1. Planning applied to Learning

While learning techniques have obtained significant success in the context of game AI, training an agent may present several challenges. For example, in complex environments such as video games, the learning process can be inefficient and slow. Furthermore, in some cases, the performance of the controllers obtained may be suboptimal. As previously mentioned, planning techniques may be used to deal with these challenges. This subsection is devoted to the presentation of some of these approaches.

5.1.1. Speeding-up Learning

Reward Shaping

Learning techniques such as RL do not make any assumptions about the environment in which they must operate or, in other words, they are not given any prior domain knowledge. However, as the state (and or action) space of the problem increases, the learning process may become much slower and even fail to learn the task within a practical time frame [21]. A reason for this is that during exploration RL may spend too much time performing suboptimal steps since it is not given any guidance on where to explore. Knowledge-based RL (KBRL) addresses this issue by incorporating domain knowledge into RL as a way to guide the learning process away from suboptimal regions of the state space and towards more promising ones in order to find the optimal policy more quickly.

Work in the field of KBRL has demonstrated that reward shaping, the process of using additional rewards as a way of providing prior knowledge to the learning process, can significantly speed-up the learning process [21]. Of particular interest to the discussion is plan-based reward shaping, an instance of KBRL in which prior knowledge is encoded as a high-level STRIPS plan used to derive the additional rewards that will help guide the learning process. Examples of applications of plan-based reward shaping include research focused on scaling RL to more complex scenarios, such as macromanagement in the RTS game StarCraft: Broodwar [21], and on improving the convergence speed and optimality (i.e., the quality of the policies obtained) of RL techniques such as Sarsa and Dyna-Q [227].

5.1.2. Improving the performance of Learning

Learning from Imitation

Although model-free DRL approaches such as DQN can achieve good results even without being given any guidance during the learning process, in many cases planning-based approaches can achieve better results, although they are, in general, orders of magnitude slower. This observation is the base

of the work presented in [228], which proposes an approach that uses UCT-based agents to generate training data offline to train a CNN to play Atari games. The rationale behind this is that the use of UCT-based agents (acting as experts playing the game) can potentially generate a dataset of gameplay traces of high quality that can be used to guide the learning process of the CNN controller through imitation learning. This can also be an alternative solution to generate these traces when human players are not available. Given the expected high quality of this dataset, the trained CNN should be able to outperform a similar controller trained without any guidance, as is the case in traditional model-free DRL. Indeed, from the three methods presented, two, namely, UCTtoClassification and UCTtoClassification-Interleaved, outperformed the results obtained by the DQN controller proposed in [229] in all of the seven Atari 2600 games tested.

Board Games

As previously discussed in Section 3, DRL has achieved great success in classical board games such as NeuroDraughts [63] in checkers and TD-Gammon [15,230] in backgammon. Both programs were trained with minimal human intervention (although hand-crafted features were used to improve the performance of both programs) via self-play (i.e., playing against other versions of themselves) using a TDL method. NeuroDraughts was able to beat quite-skilled human players, whereas TD-Gammon achieved master level play. In both cases, the incorporation of planning, in the form of a search method, proved to be beneficial in increasing the performance of the agent. In fact, in [230] the author argues that if the search level could be increased (to 3-ply in this particular case), many of TD-Gammon's technical endgame errors could possibly be eliminated. The incorporation of a search method also helped speeding-up the learning process by reducing the space of potential moves that the learning algorithm has to explore to find optimal moves.

5.2. Learning Applied to Planning

Planning techniques have obtained significant success in the context of game AI, but also present some shortcomings, particularly when faced with complex environments featuring uncertainty and partial observability, such as in video games. Apart from the fact that a model may not be readily available in such environments, some games incorporate simultaneous decision making (i.e., agents perform actions simultaneously) and can even pose constraints on the amount of time the agent is given to decide on its actions (e.g., RTS games), posing additional challenges to planning techniques. Furthermore, aspects such as uncertainty and partial observability, which characterize most video games (e.g., RTS and FPS games), may demand constant replanning in order to account for the changes occurring in the environment. These and other challenges are further discussed in the context of RTS games in [231]. As previously mentioned, learning techniques may be used to deal with these challenges. This subsection is devoted to the presentation of some of these approaches.

5.2.1. Model-Based Reinforcement Learning

As previously stated, model-based RL can be used for planning. A concrete example of this is the Imagination-Based Planner (IBP) [232], an algorithm devised to learn all the components of the planning process, more specifically, how to construct, evaluate, and execute plans, solely from experience. During this learning process, IBP is free to decide whether to act or imagine, and even how to imagine. Plans are derived by proposing imagined actions and recording the expected quality of possible sequences of these actions. When a sequence of actions (plan) with an expected high reward is encountered, IBP may decide to perform those actions on the environment (i.e., act).

At a high level, IBP works as follows: at each step a manager decides whether to act or imagine. When acting, the controller performs an action on the environment (real-world), whereas when imagining (i.e., during an imagination step), the controller proposes an imagined action that is subsequently evaluated via a model of the world. The manager can decide on the number of imagination steps to perform and also on which previous imagined or real state to imagine from.

All data generated during these steps, whether acting or imagining, is aggregated in a memory that is used to influence future decisions; see Figure 9.

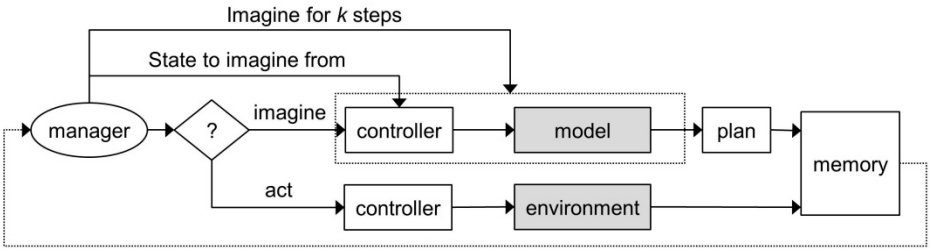

Figure 9. High level view of the Imagination-Based Planner (IBP). The manager chooses whether to act or imagine. When acting, the controller performs an action on the environment, whereas when imagining, the controller proposes an imagined action which is evaluated via a model of the world (imagination). The resulting data is aggregated in the memory.

The Collaborative Action Planning with Intention Recognition (CAPIR) [233] architecture is yet another example. The focus of CAPIR is to provide a methodology to derive NPCs capable of assisting human players in collaborative games. At a very high level, the action planning process in CAPIR is implemented as follows. During the subtask planning phase (performed offline), the main task is subdivided into subtasks encoded as MDPs. VI is then used to derive the collaboratively optimal action policy for each of these MDPs. During runtime and at each step, Bayesian inference is used to infer the subtask currently being pursued by the human player, a process referred to as intention recognition. Given this knowledge, the NPC can use the appropriate action policy learned previously to determine the most suitable actions to be carried out in order to assist the human player in the intended subtask.

5.2.2. Monte Carlo Methods

Monte Carlo methods are learning methods that learn from experience either sampled from real or simulated interaction with the environment [95]. Particle Filters (PFs) [234,235] are an example of this family of methods. PFs use a set of particles (or samples) to represent the probability density function (PDF) of the state of a stochastic system, which is iteratively refined via resampling. PFs can be used for example to solve dynamic state estimation problems, given partial or noisy observations.

Examples of application of PFs in the context of games include the Partially Observable Monte-Carlo Planning (POMCP) [236] algorithm and POMCoP (Partially Observable Monte-Carlo Cooperative Planning) [237], its variant to create NPC sidekicks in cooperative games. Both algorithms combine PFs, used to update the agent's belief state, with MCTS, used to plan based on the current belief state. PFs also allowed both algorithms to be scaled to significantly larger POMDPs (Partially Observable MDPs) where planning successfully was previously not possible. In the case of POMCoP, planning in belief space (i.e., the space of probability distributions over possible configurations of the world) also allows the AI agents (NPC sidekicks) to reason about how their own actions affect their knowledge (i.e., their beliefs) about the intentions of the human player (or collaborator).

Another example of an MC-based method is the MC Counterfactual Regret Minimization (MCCFR) algorithm. MCCFR was used to train Pluribus [238], the computer program that achieved above top human professional level play in six-player no-limit Texas hold'em poker, via self-play. MCTS (already presented in 4.1.3) is yet another example of a MC-based method.

5.2.3. Evolutionary Planning

Several EA-based approaches have been proposed to solve the planning problem, both online and offline. One such example, which draws inspiration from MCTS, is the Rolling Horizon Evolutionary Algorithm (RHEA) [239]. RHEA was proposed as a potentially faster alternative to MCTS in the context of RTS games (i.e., it can explore the search space more rapidly), where MCTS is often limited

in the extent of search it can perform due to the constraints imposed on the amount of time the player is given to decide upon which actions to perform. Similarly to MCTS rollouts, RHEA evolves a plan using a model of the environment during a specified amount of time. The first action of the evolved plan is then performed by the agent on the real environment and a new plan is evolved as before. This process, referred to as rolling horizon, is repeated until the game is over. The term rolling horizon refers to the fact that planning occurs until a limited look-ahead in the game, after which the agent must replan. RHEA has been subject to several enhancement proposals (e.g., [240,241]) and RHEA-based approaches have been used with some degree of success in the GVGAI competition [2].

Closely related to RHEA is the Online Evolutionary Planning (OEP) [242] algorithm, proposed in the context of multi-action adversarial games. OEP is more general than RHEA and proposes an approach where the plan, evolved for a complete turn, is performed to its completion as opposed to the rolling approach proposed in RHEA, where only the first action of the plan is executed and subsequently a new plan is evolved. Continual OEP (COEP) [243] proposes an enhancement to both RHEA and OEP, and continually optimizes its plans while the game is being played, taking into account new available information. This allows COEP to adapt faster to the changes occurring in the game while still performing long-term planning. COEP was tested on the macromanagement task in the RTS game StarCraft where it achieved promising results.

Other examples of EA-based planning in the context of games include the work in [244], fast random genetic search (FRGS) [245] and stochastic plan optimization (SPO) [246]. In [244], the authors augment a car racing controller with a planning module in the car racing simulator TORCS. The parameters of the planning module are evolved via EAs and a model of the track is learned and used by this planning module to derive the driving behavior of the agent interchangeably with a basic controller (also evolved via EAs). FRGS uses EAs to evolve complete plans that specify the strategies for each individual unit of an entire army using a portfolio of strategies in multi-agent planning environments applied to combat scenarios in RTS games. Finally, SPO is an offline EA-based technique devised to find and optimize plans in RTS games. Expert plans, consisting of traces of human players playing the game, annotated with the goals being pursued, are used to initialize SPO's search in plan space.

5.2.4. Deep Neural Network Architectures for Planning

Concerning the planning task, DNNs have found successful application in solving the problem of automatically learning a model of the environment, which, as previously mentioned, is essential to planning (e.g., model-based RL). A somewhat more recent line of research proposes to embed planning in the DNN architecture itself, further blurring the line between planning, DL and DRL. This subsection presents several examples of these methodologies and their different approaches to the planning problem.

Imagination-Augmented Agents

Imagination-augmented agents (I2As) [71] combine aspects from both model-free and model-based RL in order to improve data efficiency and robustness to model errors. Similarly to model-free approaches, I2As can be trained directly on low-level observations (i.e., raw images) with little domain knowledge. Moreover, I2As also use a model of the world in order to simulate imagined trajectories (or imagine). However, unlike conventional model-based methods, which usually suffer in terms of performance when dealing with imperfect models, I2As use approximate models and learn to interpret their imperfect predictions, making them more robust to model errors and misspecifications. At a high level, the I2A architecture is composed of five main components, namely, the model of the environment, the imagination core (IC), the rollout policy, the rollout encoder, and the policy module.

The model of the world, which can be either pretrained or trained conjointly with the agent, is implemented by a CNN, which takes as input the current observation and action and predicts the next observation and reward. The imagination core (IC) rolls out this model multiple times (i.e., over multiple time steps) by first initializing it with the current real observation and subsequently

with simulated observations. During this rollout phase, the actions are chosen according to the rollout policy. The resultant imagined trajectories are processed into a single result and interpreted by the rollout encoder. In other words, the rollout encoder learns to interpret this result or, even more specifically, it learns to extract useful information in terms of the agent's decision or ignore it when appropriate. Finally, the policy module combines the information extracted from the model-based predictions with the output of a model-free trajectory, obtained using real observations, and outputs the imagination-augmented policy and estimated value.

World Models

The work in [73] explores the implementation of generative neural network models of game environments (e.g., OpenAI Gym), referred to as world models. These world models can be trained quickly via unsupervised learning, using observations collected from the actual game environment, in order to learn a compressed representation of the spatial and temporal aspects of that environment. The agent can then use this world model during its learning process. A distinctive feature of this model, however, is that the agent can be entirely trained using only the model and without ever interacting with the environment as if dreaming or hallucinating via the model. The policy trained this way (i.e., by dreaming) can then be transferred to the real environment in order to solve the specified task.

At a high level, the agent's architecture proposed is composed of three main components, namely, the variational autoencoder (VAE), V; the MDN-RNN (mixture density network combined with a RNN) model, M; and the controller, C. V learns an abstract (and compressed) representation of each of the observed frames (i.e., raw images) in the form of a (compressed) latent vector z. M serves as a predictive model of the future z vectors that V is expected to produce. The controller C is responsible for choosing the actions a_t that the agent performs at each time step t, based on the concatenation of the latent vector z and M's hidden state h_t. After each action a_t, M updates its own hidden state using h_t and a_t to derive h_{t+1}; see Figure 10. Finally, the parameters of C are optimized via an EA, more specifically the Covariance-Matrix Adaptation Evolution Strategy (CMA-ES).

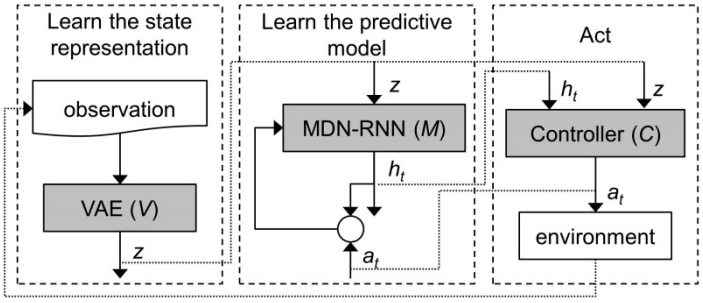

Figure 10. Flow diagram of the agent model.

Embed to Control (E2C) [247], a method for model learning and control in non-linear dynamical systems, proposes a somewhat similar approach. Similarly to world models, E2C consists of a deep generative model belonging to the family of autoencoders that can be trained in an unsupervised way, using raw pixel images. The model is derived from a control formulation in latent space, holds a belief over viable trajectories in sensory space and supports long-term prediction of image sequences or trajectories. Overall, the objective of E2C is to simplify the problem of locally optimal control in high-dimensional non-linear systems by identifying a low-dimensional latent space where this optimal control can be performed more easily.

Value Iteration Networks

The value iteration network (VIN) [248] implements a differentiable approximation of the VI planning algorithm, referred to as the VI module, which is represented as a CNN and can be trained

using standard backpropagation. This VI module is effectively embedded as a planning module within the VIN architecture and can be trained end-to-end to learn to plan. The core idea behind the VI module is that each iteration of VI can be thought of as passing the previous value function V and reward function R through a convolution layer and a max polling layer. Each channel in the convolution layer corresponds to the Q-function for a specific action. This layer is max-polled over the actions' channels to produce the next iteration value function layer V (i.e., choose the action that maximizes the Q-function). The convolution kernel weights correspond to the transition probabilities P. Given this analogy, recurrently applying a convolution layer K times is equivalent to performing K iterations of VI.

MCTSnets

MCTSnets [249] incorporate simulation-based search inside a DNN. More specifically, MCTSnets implement a neural version of MCTS (i.e., includes the same processing phases as MCTS) that learns where, what, and how to search. The idea is to maintain the desirable properties of MCTS (e.g., the use of a model and structured memory), while at the same time retaining some flexibility in order to improve some of its aspects such as the choice of nodes to expand and the statistics to store in memory and how they are propagated. In order to achieve this, MCTSnets replace every component of MCTS with a richer, learnable component.

At a high level, the computation of the network proceeds similarly to a MCTS simulation, starting from the root state and using a tree policy to choose the nodes to visit (i.e., the trajectory to traverse) until a leaf node is reached. Leaf nodes are expanded and evaluated by an embedding network and their memory vectors are initialized (i.e., the statistics gathered during search). At the end of each simulation, a backup network is used to update the memory vectors of all the nodes visited during the simulation (nodes are updated from child to parent up the trajectory). After the specified number of simulations has been performed, the memory vector of the root node (i.e., its statistics) is used to choose the action to be performed.

RNN-Based AI

RNN-Based AIs (RNNAIs) [250] are designed to learn a predictive model of the (initially unknown) environment where they must operate and then use it in order to plan and reason. RNNAIs can be trained on never-ending sequences of tasks provided either by a human user or invented by the RNNAI itself in order to improve its RNN-based world model. A distinctive feature of RNNAIs is that they learn to actively query their model in order to carry out abstract reasoning, planning and decision making (or as the author puts it, they 'learn to think').

At the core of the RNNAI architecture is the RNN-based controller/model (CM) system, used to plan and reason abstractly and efficiently (e.g., by ignoring spatio-temporal details that are irrelevant to the problem, similarly to what humans do) according to Algorithmic Information Theory (AIT) principles. At a high level, the CM system proposed works as follows. An RNN-based controller C controls the agent while it interacts with an initially unknown (and possibly partially observable) environment. This interaction history is stored and used to train an RNN-based world model M, which learns to predict new inputs from histories of previous inputs and actions. Predictive coding is used to compress the history. C may speed-up its search for rewarding behavior by learning to query and exploit the knowledge encoded in M and can also get intrinsic reward for creating experiments that improve M.

5.2.5. Planning as Inference

Planning as inference (PAI) [251] uses inference techniques to solve the planning problem (i.e., the planning problem is transformed or translated into an inference problem). At a very high level, the main idea in PAI is to use a joint probability distribution over actions, outcome states and rewards as the base to construct an internal model of the environment. The agent can then use this

model to plan, by sampling potential action-outcome trajectories from it or by assuming that its actions will yield reward (i.e., condition on reward), and then use inverse inference to discover the actions that strengthen this assumption. The work in [252] is an example of the use of PAI in the context of games.

5.2.6. MCTS Enhancements

As already mentioned in Section 4.1.3, MCTS has become a popular algorithm of choice to address a wide variety of challenges in game AI. However, MCTS in its basic form may not be suitable for all environments. Examples of this include games with high-branching factors such as turn-based multi-action adversarial games (e.g., Risk and Civilization) [253], games with sparse rewards, or games that put constraints on the time the agent is allowed to plan [254] and general video game playing, given that in this scenario MCTS may not have sufficient game-specific information to derive informative simulations (or rollouts) [255]. Another opportunity to enhance MCTS arises from the possibility of incorporating domain knowledge into the MCTS process in order to increase its performance [256].

These shortcomings and enhancement opportunities have given rise to a plethora of different methodologies, sharing in common the use of learning techniques to improve the speed, scalability, and performance of MCTS. The various enhancements proposed vary in focus and include, for example, devising tree and rollout policies that are more suitable than the originally proposed policies or finding ways to enhance those policies, implementing value functions that can substitute the simulations altogether or that can be used to stop the simulations earlier, and tuning the parameters of MCTS online.

Concerning the enhancement of the policies used in MCTS, several different approaches have been proposed. One such example is the Informed MCTS (INMCTS) [257] algorithm. INMCTS uses Bayesian models to derive models of strong players, in terms of their action probabilities, which are used by the tree policy (or rollout policy) to improve the performance of MCTS in RTS games. Another example is the work in [258], which incorporates a NN trained to predict the outcomes of games via supervised learning in the rollout policy for the early termination of simulations during rollouts in Hearthstone. Yet other approaches have used EAs to evolve rollout policies that can use information from the current game state [255,259,260] and TDL to enhance the tree policy [261] and both the tree and the rollout policies [256].

Examples concerning the remaining enhancements mentioned include the use of TDL to assign the initial values of the nodes in the search tree [256,262] and to substitute the values derived via the average of the simulations [261] and the Self-Adaptive MCTS (SA-MCTS) [177] algorithm, which proposes an EA-based approach to tune the parameters of MCTS online (i.e., UCB's exploration factor and the search depth). Yet other examples related to the shortcomings also mentioned, include the evolutionary MCTS (EMCTS) [253] algorithm, which proposes a combination of MCTS with EAs in order to improve the scalability of MCTS in games with high-branching factors (see Figure 11) and Policy Gradient for Reward Design (PGRD) [254], a method devised to learn a bonus reward function that can be used to improve the performance of UCT in gaming environments with sparse rewards or constraints in terms of the agent's planning budget.

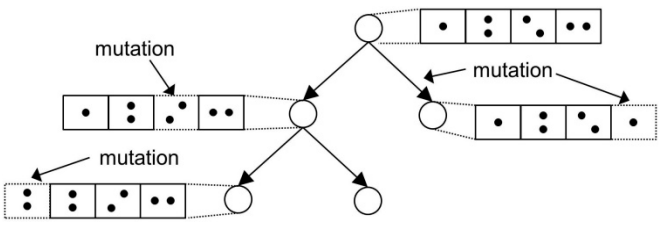

Figure 11. Tree structure of evolutionary EMCTS. Nodes represent action sequences and edges represent mutations of a single action. Repairs may be necessary if the mutations lead to invalid sequences of actions. An evaluation function is used instead of a rollout after each node expansion.

5.3. Planning and Learning

The last two sections discussed methodologies to combine planning and learning under two different perspectives, namely, the use of planning to improve learning and conversely the use of learning to improve planning. Seen through yet a different perspective, planning and learning may be combined to derive better solutions by leveraging the strengths of both and taking advantage of the fact that they are complementary. As shall be seen, several of the most recent achievements in the history of AI research have used this kind of approach. The objective of this subsection is therefore to present several methodologies devised to combine planning and learning under this third perspective.

5.3.1. Frameworks for Planning and Learning

Several different methodologies have been proposed to combine planning and learning techniques in the form of unifying frameworks. This subsection presents some of these frameworks and, in particular, those proposed in the context of game AI research.

Dyna

The Dyna [263] architecture integrates planning and learning into a single process that operates alternately on the world and on a model of the world, combining both model-free and model-based RL. More concretely, during the model-free phase (or learning phase), the agent interacts with the environment and collects real experience. This real experience is used to learn the model of the environment and to improve both the policy and the value function. During the model-based phase (or planning phase), the real experience collected is used to generate simulated experience using the model learned so far. Similarly to real-experience, simulated experience is used to improve the policy and the value function; see Figure 12. This alternated process allows Dyna to make better use of real experience and helps to speed up the learning process. Usually, the same RL algorithm is used in both the learning and the planning phases. Dyna-PI, based on the policy iteration method, and Dyna-Q, based on the Q-Learning algorithm, are two examples of this [264].

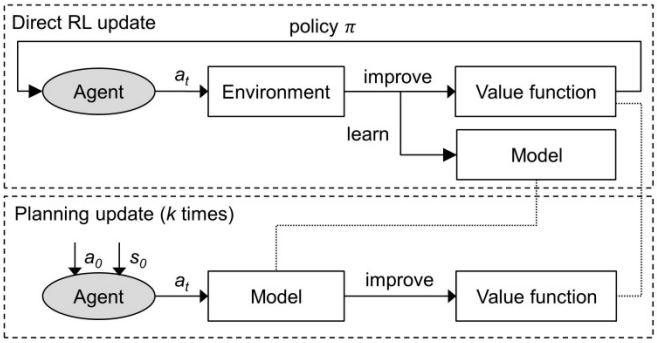

Figure 12. The general Dyna architecture. In the diagram *Direct RL update* and *Planning update* refer to the process of improving the policy and the value function via real and simulated experience, respectively. a_0 and s_0 represent the initial action and state used to initialize the planning phase.

Over time, several enhancements were proposed to improve the performance of the Dyna architecture, such as making better use of the simulated experience. Queue-Dyna [144], Prioritized Sweeping [265], and RTP-Q [266] propose different ways to achieve this. At a high level, the core idea underlying these approaches is to prioritize the value function estimate updates of the state-action pairs according to some measure of their usefulness, concerning the convergence of the learning process.

Another example, more related to games is Dyna-2 [267]. Similarly to Dyna, Dyna-2 also uses real and simulated experience. However, Dyna-2 uses two separate memories, namely, a long-term memory, learned from real experience, and a short-term memory, updated from simulated experience. The long-term memory is used to represent general knowledge about the domain (i.e., knowledge

independent from the current state), whereas the short-term memory is used to represent local knowledge (i.e., knowledge specific to the current region of the state space). During search the short-term memory is used to improve the long-term value function estimates by incorporating local knowledge. Dyna-2 was used to implement the computer Go program RLGO which achieved the highest rating among traditional ML-based programs on the 9×9 Computer Go Server.

Predictron

The predictron [268] architecture integrates planning and learning into an end-to-end training procedure. Similarly to model-based RL, the predictron uses an internal model represented by a Markov reward process (MRP), which can be rolled forward by multiple imagined planning steps to compute the value function estimates. This model is completely abstract in that is does not need to correspond to the real environment. A sufficient requirement is that the imagined trajectories generated by this abstract model produce outcomes consistent with those produced by real trajectories occurring in the real environment. However, unlike traditional model-based RL approaches, this abstract model is learned conjointly with the parameters of the value function via an end-to-end training procedure.

At a high level, the predictron is applied by rolling forward its abstract model by multiple planning steps in order to compute internal rewards, discounts, and values over multiple planning depths, which are then aggregated by an accumulator into an overall estimate of value g. These value estimates of g are used to optimize the joint parameters of the state representation, model, and value function. Concerning the accumulator just mentioned, the authors propose two different implementations, namely the *k-step* accumulator, which rolls forward its internal model k times and the λ*-weighted* accumulator, which combines many *k-step* predictron returns. Finally, taking advantage of the fact that each rollout of the predictron generates multiple value estimates g, the authors introduce consistency updates as a way to provide yet an additional signal to the learning process. Implementation-wise, the predictron consists of a DNN with an MRP as a recurrent core that is unrolled multiple times.

Temporal-Difference Search

Temporal-difference search (TD search) [269] proposes an approach to high-performance search in MDPs and two-player games that combines TDL with simulation-based search. TD search tries to solve two shortcomings of MCTS, namely, the high variance of the estimates produced by the MC simulations and the independent manner in which the states are evaluated (i.e., there is no generalization between similar states). Similarly to MCTS, TD search uses simulated experience, derived from a model of the environment, to update the value function. However, contrary to MCTS, instead of using a search tree, TD search uses value function approximation and, instead of using MC evaluation, TD search uses TDL. On one hand using value function approximation means that the outcome of a single simulation, starting from state s, can be used to update the value function of similar states to s (i.e., generalization between related states). Furthermore, this value function can approximate the value of states not yet visited. On the other hand, by using TDL the value function can bootstrap from previous values, which often translates into a reduction in the variance of its estimates. TD search was tested on the 9×9 Computer Go Server, where it achieved very good results.

Temporal-Difference Tree Search

Temporal-difference tree search (TDTS) [270] combines RL and MCTS methods into a unifying framework and family of algorithms that can be understood both as a generalization of MCTS with RL mechanics, more specifically TDL, and as an extension of RL methods, more specifically TD methods, with MCTS concepts. From RL, TDTS introduces concepts such as first visit updates and bootstrapping TD backups to replace MC backups, similarly to TD search. In fact, the two frameworks are closely related. TD search is, however, more specific to Go, whereas TDTS intends to be more general. From MCTS, TDTS introduces several concepts such as the incremental representation of the state space, a mechanism by which only the parts of the state space considered to be more relevant

are represented or kept in memory at each time, as opposed to the whole state space, as in traditional value function approximation techniques.

Other MCTS concepts introduced include a representation policy, which defines how the representation model of the state space is adapted online, a playout value function, used to replace the missing value estimates of the non-memorized (or non-represented) parts of the state space with playout values, and a playout policy that derives its decisions based on playout values. Together, these mechanisms can be used to recreate the four phases of MCTS. Overall, TDTS encompasses online and offline backups, first-visit and every-visit updates, and on-policy and off-policy control. An example of an instance of a TDTS method is Sarsa-UCT(λ) [270], an algorithm that has achieved strong results in the GVGAI competition.

The Options Framework

The options framework [271] proposes a compact and unified framework for temporal abstraction in the context of RL and MDPs. The core idea is to extend the usual notion of action in RL to include options, a generalization of primitive actions that includes temporally extended courses of action (i.e., actions that might result in state transitions of extended and variable duration). An option $(\mathcal{J}, \pi, \beta)$ consists of three components: an initiation set $\mathcal{J} \subseteq S$, a policy $\pi : S \times A \rightarrow [0, 1]$, and a termination condition $\beta : S^+ \rightarrow [0, 1]$. An option is available in state s_t if $s_t \in \mathcal{J}$. If the option is taken, actions are selected according to π until the option terminates according to β, in which case the agent has the opportunity to select another option.

Examples of options include high level actions such as picking an object, as well as primitive actions such as joint torques. By integrating temporal abstraction into the MDP framework (via options), both the planning and learning processes may take advantage of the simplifications and efficiency gains potentially found when working at higher levels of temporal abstraction. The option-critic architecture [272] proposes an approach to automatically learn options without the need to provide additional rewards or subgoals. More specifically, option-critic learns the internal policies π and the termination functions β of the options simultaneously with the policy over options (the policy used to choose the options).

Model-Based Reflection and Self-Adaptation

Techniques for model-based introspection (or reflection) endow an agent with a model of its reasoning processes so that the agent can use this model to analyze its own actions and experiences and make proper modifications to its knowledge and reasoning (i.e., self-adapt). The Reflective Evolutionary Mind (REM) [273] is an example of a model-based reflection technique that combines model-based reflection with generative planning and RL. REM uses the Task Method Knowledge Language (TMKL) to encode the agent in terms of its reasoning architecture, including the tasks that the agent can perform and the knowledge used to accomplish them.

At a high level, REM uses the TMKL model of the agent and the trace of its actions (i.e., the internal processing of the agent while performing a task) to localize the modifications required to the model of the agent in order to address a failure occurring while executing a task or to enable the agent to accomplish a new task. Once a modification is located, it is executed via one of the general adaptation methods available in REM, namely, failure-driven adaptation, generative planning and RL. Model-based reflection techniques have been used in the context of games, for example, to train AI agents to play multi-player games in partially observable worlds [274].

Cognitive Architectures

The objective of artificial cognitive architectures is to integrate many of the aspects related to human cognition, such as emotion, planning, learning, and attention mechanisms, into effective control systems [275]. The CERA-CRANIUM [276] cognitive architecture is an example of this family of architectures. At a high level, CERA (Conscious and Emotional Reasoning Architecture) is a control

architecture structured in layers following a subsumption relation (i.e., higher layers process more abstract content than lower layers), whereas CRANIUM provides a tool to create and manage parallel processes. Concerning their application in games, CERA-CRANIUM was used to derive believable or human-like AI agents in the game Unreal Tournament 2004 [275], whereas CERA was used to test the cognitive skills of CERA-controlled agents in the game Unreal Tournament 3 [277]).

Yet another example is spiking neural networks (SNNs) [278]. SNNs try to mimic natural neural networks (e.g., the human brain) more closely than other DNN architectures by incorporating computational methods analogous to the natural mechanisms found in those NNs, such as neuronal and synaptic states. SNNs also incorporate the concept of time in their computational model. Overall, the mathematical model of spiking neurons intends to describe the output of biological neurons in a more realistic way. Examples of application of SNNs in games include the automatic construction of environmental states from image inputs in order to derive an agent that can play the side-scroller game Flappy Bird [279] and goal-directed planning [280].

Cognitive Modeling

Cognitive models govern what the agent knows, how that knowledge is acquired, and how it can be used to plan actions, and therefore surpass behavioral models. The Cognitive Modeling Language (CML) [281] was devised to help creating these cognitive models in applications such as advanced character animation and automated cinematography. CML can be used to integrate domain knowledge, specified in terms of actions, their preconditions, and effects, into the agent and then direct its behavior in terms of goals. This in turn can prove to be a very useful game design support tool, since specifying the behaviors of agents augmented with cognitive skills can be done at a higher level of abstraction that is more natural and intuitive to the game designer or animator. As an example of this potential utility, given a behavior outline specified by the animator, the agent may automatically find a sequence of actions satisfying the specification outlined through reasoning.

Other Frameworks

In addition to the frameworks just mentioned, several others were proposed in the context of games. The Lazy Explanation-Based Learning (LEBL) [282] framework is one such example. LEBL is a learning technique devised to learn planning knowledge in two-player games, using incomplete explanations produced by considering only a subset of the set of possible actions or moves at each state during the expansion of the state space. Another example is the work in [283], which proposes an architecture combining the reasoning capabilities of classical planning (i.e., the ability to reason over long-term goals and devise plans to achieve them) with the reactive control capabilities of NNs (to execute those plans). Yet another example is the PEORL [225] framework, which proposes a methodology to address the problem of decision making in dynamic environments with uncertainty, based on the integration of symbolic planning, to guide the learning process, with hierarchical RL, to enrich symbolic knowledge and improve planning.

Yet other frameworks devised to integrate planning and learning in more general contexts other than games include Prodigy [284], an architecture that integrates planning and learning via mutually interpretable knowledge structures, PELEA [223,224], devised to dynamically integrate a wide range of state-of-the-art components for planning, execution, monitoring, replanning, and learning, and the Planning Execution and Learning Architecture (PELA) [285], formulated to use off-the-shelf interchangeable planning and learning components to solve planning tasks involving uncertainty.

5.3.2. Board Games

As already mentioned, both planning and learning techniques have achieved great success in classical board games. Despite this, both techniques present some challenges when used in the context of this game genre. In the case of planning, the success and feasibility of search techniques relies heavily on domain-specific knowledge to derive good heuristics to simplify the search space

and to choose features to represent the board positions and construct their evaluation functions. Furthermore, the performance of these search algorithms can be greatly hindered when tackling games with uncertainty, such as backgammon, given the limited depth of the search that can be feasibly achieved [12,230]. In the case of learning, the lack of a look-ahead mechanism may introduce variability in the quality of the performance of the program throughout the course of the game (e.g., poorer during endgames), such as in the case of TD-Gammon [230].

This fact led many researchers to combine techniques from both research fields to tackle these challenges. Examples of this include work in Shogi [286], KnightCap [13] and Meep [14] in chess, Logistello [11] in Othello, TD-Gammon [15] in backgammon, LS-VisionDraughts [81] in checkers, and AlphaGo [17] and [262] in Go. Some of these approaches were, in fact, quite successful and achieved master-level (e.g., KnightCap, Meep, TD-Gammon) and even human world-champion level play (e.g., Logistello, AlphaGo). Generally, learning is used to learn the representation of the board positions and their evaluation function, whereas planning is used as a look-ahead mechanism to help choosing the moves.

In AlphaGo, for example, both a fast rollout policy network p_π and a policy network p_σ are first trained via supervised learning, using game traces of human experts. Next, using games of self-play, a new policy network p_ρ, initialized from p_σ, is further refined (i.e., trained) via policy gradient learning. This in turn generates a new dataset that is used to train a value network v_θ. All of these networks are CNNs that automatically learn and extract the appropriate representation of the board positions (i.e., features) from raw images. Finally, during the MCTS simulations, p_ρ is used to derive the prior probabilities for each action during the expansion phase of each node, whereas p_π is used as the rollout policy to choose the actions during the simulation phase. In practice, p_σ was used instead of p_ρ because it obtained better results. v_θ is used to evaluate the leaf nodes during the backup phase. AlphaGo Zero [287] further extended this work and relies only on RL without the use of any human data or domain knowledge, apart from the game rules. When tested against each other, AlphaGo Zero convincingly defeated AlphaGo.

Another interesting line of research work is Expert Iteration (ExIt) [288], a novel RL algorithm that has shown great promise by defeating MoHex 1.0, a recent Olympiad Champion, in the board game Hex. In ExIt, a tree search algorithm, acting as the expert, is used to assist training a DNN, acting as the apprentice, in a fashion similar to imitation learning. The DNN in turn is used to help improve the performance of the tree search algorithm during the so-called expert improvement step, where the DNN can be used as an initial bias to guide the search or to help estimate the values of the states in the search tree, or both. In practice, this process creates a closed learning loop in which the apprentice and the expert keep improving each other.

5.3.3. General Video Game Playing

As previously mentioned, GVGP is a challenging research area in which the goal is to devise general-purpose video game playing agents. While approaches composed of a single technique (e.g., MCTS and its variants) continue to be quite successful, as proved by the results of the GVGAI competition presented in [2] (e.g., the winning entries of the CIG-14, CIG-16, and FDG-18 2-player track legs of the competition), hybrid approaches, such as those featuring the combination of planning and learning techniques, have also been proposed, with varying degrees of success. One example of such an approach, which has achieved a lot of success in the context of the GVGAI competition, is the ToVo2 controller [289], winner of the 2016 World Congress on Computational Intelligence (WCCI) 2-player track and the 2017 Congress on Evolutionary Computation (CEC-17) 2-player track legs of the competition. ToVo2 combines MCTS with Sarsa-UCT(λ) [270] in order to take advantage of the generalization of UCT and the learning capabilities of Sarsa.

SA-MCTS [177] is another example of this type of hybrid approach. SA-MCTS is an algorithm devised to tune the parameters of MCTS online (i.e., UCB's exploration factor and the search depth), using EA-based allocation strategies. Another example is the work in [240], which proposes several

enhancements to RHEA, including an EA-tree enhancement by which a statistical tree is maintained alongside the population evolved by RHEA and used at the end of evolution to choose the move to make (i.e., with the highest UCB value). Yet another example is the work in [290], which leverages EAs to evolve tree policies for MCTS, one for each of the games used in the study, to try to take advantage of the characteristics specific to each game. A final example is TD-MCTS [291], an algorithm that combines True Online Sarsa(λ) [292] and MCTS. True Online Sarsa is used as the rollout policy of MCTS in order to take advantage of past experience and the domain knowledge automatically acquired during the learning process.

Other relevant research in the field of GVGP includes the AlphaZero algorithm [293] and the more recent MuZero algorithm [294]. AlphaZero builds upon and generalizes AlphaGo Zero [287] to a broader range of classical board games that include Go, chess, and shogi, by combining a general-purpose DRL algorithm with a general-purpose search technique (i.e., MCTS) into a powerful algorithm that can learn from scratch through self-play and without using domain-specific knowledge. AlphaZero achieved astonishing results by convincingly defeating the 2016 TCEC season 9 world champion (Stockfish) in chess, the 2017 CSA world champion (Elmo) in shogi, and the previously published version of AlphaGo Zero in Go. MuZero extends AlphaZero to the Atari 2600 gaming platform, where it was able to achieve a new state of the art (57 games from the Atari 2600 gaming platform were used for this evaluation). MuZero was also able to match the performance of AlphaZero in chess, shogi and Go, without being given any knowledge about the rules of these games.

5.3.4. Game Design Support

One of the tasks related to game design is playtesting. Playtesting consists of testing a game by playing it in order to assess it under several different aspects such as whether it actually works as intended or if it is fun or aesthetically pleasing. Playtesting is also a costly part of the game design process [295]. While some of the aspects just mentioned must be performed by human playtesters (e.g., determine whether the game is aesthetically pleasant), other aspects may be automated (e.g., test game levels). One approach that intends to address this issue is Monster Carlo 2 (MC2) [295]. MC2 combines the learning capabilities of Unity's Machine Learning Agents Toolkit with the tree search capabilities of Monster Carlo [296], a search-based playtesting framework for Unity games, into an automated playtesting system intended for independent game developers with or without ML expertise. By combining planning and learning, MC2 can train agents using either gameplay traces from human experts or via self-play simulations using MCTS, depending on whether these gameplay traces are available or not. By providing alternative ways to train the agent, MC2 also offers a solution to prevent unnatural super-human play, resulting from MCTS exploits.

Path finding is yet another example. Real-time heuristic search algorithms are good candidates to implement path finding in video games (e.g., planning time can be controlled), however, their performance depends greatly on the search space [297]. This means that developers would have to test all of the algorithms in order to find the most appropriate one to solve the search problem, given a map. The work in [297] proposes an approach to automate this process. More specifically, the authors train a DL classifier to choose the best planning algorithm (from a portfolio of algorithms) to solve the path finding problem on new yet unseen maps without the need to evaluate the algorithms on those maps.

5.3.5. More Examples of Planning and Learning in Video Games

Besides the approaches already mentioned, planning and learning have been combined to address a plethora of challenges in a wide variety of video game genres.

Control

Planning and learning have been combined to derive control solutions in several different game genres. An example of this is the work in [298], where the authors propose an approach to solve levels of the physics-based platform game GF. The task of solving a level of GF is subdivided into

the task of solving an individual platform (SP1), deciding on which platform to solve next (SP2), and moving between platforms (SP3). RL is leveraged to solve SP1 and SP3, whereas a depth-first search is used to solve SP2. The work in [299] is another example. In this work, RL is used to derive macro (i.e., the choice of aiming path) and micro (i.e., how to shoot) actions, which are used to guide a tree search algorithm during the planning phase to choose the appropriate macro and micro actions according to the specific game situation in nine-ball, a billiards game. Opponent interception in FPS games is another example. In [300], the authors use inverse RL (IRL) to learn the motion models of the players from past observations during gameplay, which are combined with a navigational map to generate a set of likely paths for the opponent. These paths are used to seed the motion models of a set of particle filters. A planner is then used to create an interception plan based on the output of the particle filters. Yet more examples include combining DL, for high level strategy selection, and tree search, to improve the tactical actions of the units, in RTS games [301], and leveraging ML to cluster table states in order to speed-up the look-ahead search used to plan the shots in CueCard, the winner of the 2008 Computer Olympiad computational pool tournament [302].

Multi-Agent Environments

Cerberus [303], a machine learning framework for team-based capture-the-flag games, is an example of work concerning the combination of planning and learning in the context of multi-agent environments. In Cerberus, RL is used to derive the high-level behavior of the AI agents (i.e., team behavior), whereas NNs, trained offline via imitation learning from a human trainer, provide their low-level behavior (i.e., individual behavior). A* is used for path planning (e.g., to plan paths that avoid allies or enemies or to perform cooperative attacks).

5.3.6. Competitions for Planning and Learning Research

Several competitions have been proposed for planning and learning research in the context of games. Examples of these competitions include the Geometry Friends Game AI competition [4], held at CoG (former CIG), and the General Video Game AI competition [2], held at several different conferences (e.g., FDG, WCCI, CoG).

The GF competition is divided into two tracks, namely the Cooperative track and the Single AI track, which in turn is subdivided into two subtracks, the Circle track and the Rectangle track. The focus of the Cooperative Track is the development of agents that can cooperate in order to solve the levels of the game. The Circle track and the Rectangle track are devoted to the development of an agent to control the circle character and the rectangle character respectively. The competition is based on the GF game, a cooperative physics-based puzzle game where two agents, namely a circle and a rectangle, featuring different skills, are used to solve the levels of the game (i.e., collect all the diamonds scattered around the multiple platforms in the level). The purpose of these different agents is to foster different types of research. The circle agent presents more challenges in terms of movement control (e.g., timing jumps) and is usually more amenable to learning techniques, whereas the rectangle agent presents more challenges in terms of solving the puzzle (e.g., determine the order to collect the diamonds) and is therefore more amenable to planning techniques. Examples of work proposed in the context of the GF competition can be found in [4,298].

The goal of the GVGAI competition, as previously mentioned, is to foster the implementation of general game playing agents that can play well in a variety of different games, without human intervention. The competition is divided into Game Playing tracks and PCG tracks. More specifically, the Game Playing Tracks consist of the Single-Player and Two-Player Planning tracks and the Single-Player Learning track, whereas the PCG tracks consist of the Level Generation track and the Rule Generation track. While the single-player planning track and the single-player learning track promote research on planning and learning techniques, respectively, the two-player planning track introduces opponent modeling and interaction with other agents in the game as additional challenges, presenting further opportunities to combine planning and learning. Furthermore, the set of games

used in the competition is chosen in order to span a wide spectrum of technical challenges. In this way, some games, such as cooperative games, may favor some techniques (e.g., MCTS), whereas other games, such as puzzle games, may favor yet different techniques (e.g., EAs) [2]. Examples of work proposed in the context of the GVGAI competition can be found in [2,289].

5.4. Discussion

The previous subsections presented methodologies to combine planning and learning from three different points of view. While the first two concerned the use of one paradigm to improve the other, namely, planning applied to improve learning and learning applied to improve planning, the last presented methodologies to combine the strengths of both. Given this, some interesting remarks are noteworthy. One such remark is that several learning techniques can be used for planning (e.g., EAs, DNNs, DRL). In the case of DNNs, some of the approaches presented are even starting to incorporate planning into the architecture of the DNN itself, further blurring the line between learning and planning.

Another interesting remark concerns the wealth and diversity of research dedicated to the enhancement of MCTS via several learning techniques such as DRL and EAs. This again seems to prove the huge popularity and widespread application of MCTS in games. Furthermore, other techniques such as DNNs are taking inspiration from MCTS in order to derive novel solutions (e.g., MCTSnets). AP techniques, on the other hand, as already pointed out in Sections 4.1.4 and 4.2, continue to be less popular, hinting at the need of further research work concerning their integration with learning and, more specifically, regarding the implementation of methods, tools, and algorithms to make this integration easier and more attractive.

Yet another interesting observation concerns the plethora of frameworks that have been proposed in the literature to integrate planning and learning. Not only do these frameworks present very different approaches to achieve this, they also incorporate different influences from other research fields such as neurobiology and cognitive science (e.g., SNNs). A final remark concerns the astonishing success achieved by the combination of DRL and MCTS, an approach that has played a major role in some of the most recent milestones in AI research, such as AlphaGo, AlphaGo Zero, AlphaZero, and MuZero.

Overall, the combination of planning and learning, concerning the three points of view presented, has achieved significant success in game AI. Nevertheless, there are still some opportunities for improvement. One such opportunity was discussed in Section 4.2 and concerns the need to devise easier ways of combining AP with learning techniques, given that these planning techniques allow the use of more powerful reasoning approaches, such as first-order-logic. Another opportunity for improvement concerns general video game playing. While some hybrid approaches combining planning and learning have been proposed with some success in this domain, approaches consisting of a single technique such as MCTS continue to be quite popular and successful, hinting at the fact that hybrid approaches can be further improved.

Yet another concern relates to the hardware requirements of some of these approaches. In the case of AlphaGo, the final version used a parallelized architecture of over 48 CPUs and 8 GPUs, and training of, for example, the policy network p_σ, took around 3 weeks on 50 GPUs. While hardware improvements can undoubtedly mitigate these concerns, devising methodologies to further fine tune these algorithms in terms of data efficiency and speed of the learning process may present itself as yet another opportunity for further improvements. Table 5 presents some examples of combinations of techniques from planning and learning and related work.

Table 5. Combinations of planning and learning techniques and examples of related work. In the table the following terms were abbreviated as: classical planning (CP), Alpha-Beta (α-β), depth-first search (DFS), Bayesian model (BM), heuristic search (HS), look-ahead search (LAS), and Wavefront Planner (WFP).

Techniques	Examples of Related Work
PF, MCTS	POMCP [236], POMCoP [237]
DL, MCTS	[228,258,301]
DRL, CP	[21,283]
EA, Minimax	Anaconda [80]
EA, α-β	Blondie25 [79]
DRL, Minimax	NeuroDraughts [63], KnightCap [13]
DRL, EA, α-β	LS-VisionDraughts [81]
DRL, HS	TD-Gammon [15,230]
BM, MCTS	INMCTS [257]
EA, MCTS	SA-MCTS [177], EMCTS [253], [240,255,259,260,290]
DRL, MCTS	PGRD [254], TDTS [270], AlphaGo [17], AlphaGo Zero [287], ExIt [288], TD-MCTS [291], AlphaZero [293], MuZero [294], MC2 [295], [256,261,299]
DRL, A*	Cerberus [303]
RL, DFS	[298]
IRL, PF, WFP	[300]
RL, α-β	Meep [14]
ML, α-β	Logistello [11,52]
ML, LAS	CueCard [302]
RL, UCT	[262]
RL, Minimax	[286]

6. Conclusions

The purpose of this work was to present a survey of planning and learning in game AI, with a particular focus on the many methodologies proposed to combine both techniques. The paper started by presenting some of the most relevant challenges in game AI as a cursory introduction to this research field. Next, the most well-known learning and planning techniques were presented. Each technique was briefly introduced in terms of its theoretical foundations and applications in games. The perspective of the gaming industry, regarding the adoption of these techniques, was also briefly discussed. The purpose of this discussion was to serve as an introduction to learning and planning, both as two separate subfields of AI with distinct purposes and featuring different approaches, and as standalone techniques that have achieved great success and application in games.

The paper then presented multiple methodologies proposed to combine planning and learning. The discussion was structured from three different points of view, namely, the application of planning techniques to improve learning; conversely, the application of learning techniques in order to improve planning; and, lastly, the combination of both techniques, leveraging their individual strengths and complementary relationship to derive better solutions. The many open challenges and possible future research directions concerning these research fields were also briefly presented and discussed. The paper concludes with some additional remarks concerning what to be expected in the future, regarding planning and learning research in games.

6.1. Research

It is expectable that new algorithms, methods, and frameworks will continue to be developed in academia concerning research on planning and learning in games, with particular focus on hybrid techniques integrating DL with other learning and planning techniques. It is also foreseeable that DL, DRL, and EAs in learning research and MCTS in planning research will continue to be the most popular (and dominant) techniques in the near future. It also seems that academia has begun to favor

learning techniques as their main approach to implement planning. This is another research topic that will most likely gain significant attention.

6.2. Research Topics

Concerning the existing research areas, it is expectable that topics such as Transfer Learning, Adaptive Game AI, and General Video Game Playing will continue to be further improved. MuZero is an example of recent work in GVGP and other examples will likely follow. Regarding newer research topics, it is to be expected that academia will continue to push the state of the art in the domain of E-sports, RTS games, multiplayer online games, and virtual, augmented and mixed reality-based games, which begin to have a more noticeable presence in the video game market.

6.3. Academia and the Gaming Industry

Finally, it can be expected that academia and the gaming industry will begin to approach development more closely, so that the industry can profit from the astonishing accomplishments derived in academia. This may, however, be a slow process. On one hand, academia will need time to implement mechanisms that allow these techniques to be more easily programmed, debugged, fine-tuned, and interpretable by their human programmers, allowing greater authorial control over the artificial agents created that use them. On the other hand, the industry will need time to adapt and master this new ecosystem of technical solutions to create more immersive and realistic game AI.

Author Contributions: Conceptualization, F.F.D. and N.L.; methodology, F.F.D.; investigation, F.F.D.; writing—original draft preparation, F.F.D.; writing—review and editing, F.F.D.; supervision, N.L., A.P. and L.P.R.; project administration, N.L., A.P. and L.P.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Fundação para a Ciência e a Tecnologia, grant number SFRH/BD/145723/2019 - UID/CEC/00127/2019.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rohlfshagen, P.; Liu, J.; Perez-Liebana, D.; Lucas, S.M. Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game. *IEEE Trans. Games* **2018**, *10*, 233–256. [CrossRef]
2. Perez-Liebana, D.; Liu, J.; Khalifa, A.; Gaina, R.D.; Togelius, J.; Lucas, S.M. General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms. *IEEE Trans. Games* **2019**, *11*, 195–214. [CrossRef]
3. Genesereth, M.; Björnsson, Y. The International General Game Playing Competition. *AI Mag.* **2013**, *34*, 107–111. [CrossRef]
4. Prada, R.; Lopes, P.; Catarino, J.; Quiterio, J.; Melo, F.S. The Geometry Friends Game AI Competition. In Proceedings of the Proceedings of the IEEE Conference on Computational Intelligence and Games, Tainan, Taiwan, 31 August–2 September 2015; pp. 431–438. [CrossRef]
5. Gomez, F.; Miikkulainen, R. Incremental Evolution of Complex General Behavior. *Adapt. Behav.* **1997**, *5*, 317–342. [CrossRef]
6. Asensio, J.M.L.; Donate, J.P.; Cortez, P. Evolving Artificial Neural Networks Applied to Generate Virtual Characters. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 23–27. [CrossRef]
7. Mahmoud, I.M.; Li, L.; Wloka, D.; Ali, M.Z. Believable NPCs In Serious Games: HTN Planning Approach Based on Visual Perception. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 248–255. [CrossRef]
8. Torrey, L. Crowd Simulation Via Multi-Agent Reinforcement Learning. In Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 11–13 October 2010; pp. 89–94.
9. Campbell, M.; Hoane, A.J.; Hsu, F.H. Deep Blue. *Artif. Intell.* **2002**, *134*, 57–83. [CrossRef]

10. Schaeffer, J.; Lake, R.; Lu, P.; Bryant, M. CHINOOK The World Man-Machine Checkers Champion. *AI Mag.* **1996**, *17*, 21–29. [CrossRef]
11. Buro, M. From Simple Features to Sophisticated Evaluation Functions. In Proceedings of the International Conference on Computers and Games, Tsukuba, Japan, 11–12 November 1998; Springer: Berlin/Heidelberg, Germany; pp. 126–145. [CrossRef]
12. Russell, S.J.; Norvig, P. *Artificial Intelligence A Modern Approach*; Prentice Hall: New Jersey, NJ, USA, 2010; p. 1132. ISBN 978-0-13-604259-4.
13. Baxter, J.; Tridgell, A.; Weaver, L. Learning to Play Chess Using Temporal Differences. *Mach. Learn.* **2000**, *40*, 243–263. [CrossRef]
14. Veness, J.; Silver, D.; Uther, W.; Blair, A. Bootstrapping from Game Tree Search. In Proceedings of the 22nd International Conference on Neural Information Processing Systems, Vancouver, Canada, 7–10 December 2009; pp. 1937–1945.
15. Tesauro, G. TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Comput.* **1994**, *6*, 215–219. [CrossRef]
16. 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* **2015**, *518*, 529–533. [CrossRef] [PubMed]
17. 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* **2016**, *529*, 484–489. [CrossRef]
18. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Dębiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv* **2019**, arXiv:1912.06680.
19. Arulkumaran, K.; Cully, A.; Togelius, J. Alphastar: An evolutionary computation perspective. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, 13–17 July 2019; pp. 314–315. [CrossRef]
20. Jiménez, S.; La Rosa, T.D.; Fernández, S.; Fernández, F.; Borrajo, D. A Review of Machine Learning for Automated Planning. *Knowl. Eng. Rev.* **2012**, *27*, 433–467. [CrossRef]
21. Efthymiadis, K.; Kudenko, D. Using Plan-Based Reward Shaping To Learn Strategies in StarCraft: Broodwar. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Niagara Falls, ON, Canada, 11–13 August 2013; pp. 407–414. [CrossRef]
22. Partalas, I.; Vrakas, D.; Vlahavas, I. Reinforcement Learning and Automated Planning: A Survey. In *Artificial Intelligence for Advanced Problem Solving Techniques*; Vlahavas, I., Vrakas, D., Eds.; IGI Global: Hershey, PA, USA, 2008; pp. 148–165. [CrossRef]
23. Nagle, A.; Wolf, P.; Riener, R. Towards a system of customized video game mechanics based on player personality: Relating the Big Five personality traits with difficulty adaptation in a first-person shooter game. *Entertain. Comput.* **2016**, *13*, 10–24. [CrossRef]
24. Missura, O.; Gärtner, T. Player Modeling for Intelligent Difficulty Adjustment. In Proceedings of the International Conference on Discovery Science, Porto, Portugal, 3–5 October 2009; Gama, J., Santos Costa, V., Jorge, A., Brazdil, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 197–211. [CrossRef]
25. Kaidan, M.; Harada, T.; Chu, C.Y.; Thawonmas, R. Procedural Generation of Angry Birds Levels with Adjustable Difficulty. In Proceedings of the IEEE Congress on Evolutionary Computation, Vancouver, Canada, 24–29 July 2016; pp. 1311–1316. [CrossRef]
26. Hao, Y.; He, S.; Wang, J.; Liu, X.; Yang, J.; Huang, W. Dynamic Difficulty Adjustment of Game AI by MCTS for the Game Pac-Man. In Proceedings of the Sixth International Conference on Natural Computation, Yantai, China, 10–12 August 2010; pp. 3918–3922. [CrossRef]
27. Kim, H.T.; Kim, K.J. Learning to Recommend Game Contents for Real-Time Strategy Garners. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 184–191. [CrossRef]
28. Justesen, N.; Risi, S. Learning Macromanagement in StarCraft from Replays using Deep Learning. In Proceedings of the IEEE Conference on Computational Intelligence and Games, New York, NY, USA, 22–25 August 2017; pp. 162–169. [CrossRef]

29. Weber, B.G.; Mateas, M. A Data Mining Approach to Strategy Prediction. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Milano, Italy, 7–10 September 2009; pp. 140–147. [CrossRef]
30. Yamamoto, K.; Mizuno, S.; Chu, C.Y.; Thawonmas, R. Deduction of Fighting-Game Countermeasures Using the k-Nearest Neighbor Algorithm and a Game Simulator. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 437–441. [CrossRef]
31. Kristo, T.; Maulidevi, N.U. Deduction of Fighting Game Countermeasures using Neuroevolution of Augmenting Topologies. In Proceedings of the International Conference on Data and Software Engineering, Denpasar, Indonesia, 26–27 October 2016; pp. 148–154. [CrossRef]
32. Valls-Vargas, J.; Ontañón, S.; Zhu, J. Exploring Player Trace Segmentation for Dynamic Play Style Prediction. In Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Santa Cruz, CA, USA, 15–18 November 2015; pp. 93–99.
33. Mahlmann, T.; Drachen, A.; Togelius, J.; Canossa, A.; Yannakakis, G.N. Predicting Player Behavior in Tomb Raider: Underworld. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Dublin, Ireland, 18–21 August 2010; pp. 178–185. [CrossRef]
34. Chen, Z.; El-nasr, M.S.; Canossa, A.; Badler, J.; Tignor, S.; Colvin, R. Modeling Individual Differences through Frequent Pattern Mining on Role-Playing Game Actions. In Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AAAI Technical Report WS-15-23, Santa Cruz, CA, USA, 15–18 November 2015; pp. 2–7.
35. Yannakakis, G.N.; Maragoudakis, M.; Hallam, J. Preference Learning for Cognitive Modeling: A Case Study on Entertainment Preferences. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2009**, *39*, 1165–1175. [CrossRef]
36. Packard, B.; Ontañón, S. Policies for Active Learning from Demonstration. In Proceedings of the AAAI Spring Symposium on Learning from Observation of Humans Technical Report SS-17-06, Palo Alto, CA, USA, 27–29 March 2017; pp. 513–519.
37. Packard, B.; Ontañón, S. Learning Behavior from Limited Demonstrations in the Context of Games. In Proceedings of the 31st International Florida Artificial Intelligence Research Society Conference, Melbourne, FL, USA, 21–23 May 2018; pp. 86–91.
38. Ross, S.; Gordon, G.J.; Bagnell, J.A. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 11–13 April 2011; PMLR; pp. 627–635.
39. Sharma, M.; Holmes, M.; Santamaria, J.; Irani, A.; Isbell, C.; Ram, A. Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007; pp. 1041–1046.
40. Taylor, M.E.; Stone, P. Transfer Learning for Reinforcement Learning Domains: A Survey. *J. Mach. Learn. Res.* **2009**, *10*, 1633–1685.
41. Ferreira, L.; Toledo, C. A Search-based Approach for Generating Angry Birds Levels. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 413–420. [CrossRef]
42. Dormans, J.; Bakkes, S. Generating Missions and Spaces for Adaptable Play Experiences. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 216–228. [CrossRef]
43. Hy, R.L.; Arrigoni, A.; Bessiere, P.; Lebeltel, O. Teaching Bayesian Behaviours to Video Game Characters. *Rob. Auton. Syst.* **2004**, *47*, 177–185. [CrossRef]
44. García-Sánchez, P.; Tonda, A.; Mora, A.M.; Squillero, G.; Merelo, J.J. Automated playtesting in collectible card games using evolutionary algorithms: A case study in hearthstone. *Knowl. -Based Syst.* **2018**, *153*, 133–146. [CrossRef]
45. Liapis, A.; Holmgård, C.; Yannakakis, G.N.; Togelius, J. Procedural Personas as Critics for Dungeon Generation. In Proceedings of the 18th European Conference on the Applications of Evolutionary Computation, Copenhagen, Denmark, 8–10 April 2015; Springer: Cham, Switzerland, 2015; pp. 331–343. [CrossRef]
46. Ponsen, M.; Muñoz-Avila, H.; Spronck, P.; Aha, D.W. Automatically Generating Game Tactics through Evolutionary Learning. *AI Mag.* **2006**, *27*, 75–84. [CrossRef]
47. Martínez-Arellano, G.; Cant, R.; Woods, D. Creating AI Characters for Fighting Games using Genetic Programming. *IEEE Trans. Comput. Intell. AI Games* **2017**, *9*, 423–434. [CrossRef]

48. Maissiat, J.; Meneguzzi, F. Adaptive High-Level Strategy Learning in StarCraft. In Proceedings of the 12th Brazilian Symposium on Games and Digital Entertainment, São Paulo, Brazil, 16–18 October 2013; pp. 17–24.
49. Denzinger, J.; Winder, C. Combining coaching and learning to create cooperative character behavior. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Colchester, UK, 4–6 April 2005; pp. 78–85.
50. Sharifi, A.A.; Zhao, R.; Szafron, D. Learning Companion Behaviors Using Reinforcement Learning in Games. In Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Palo Alto, CA, USA, 11–13 October 2010; pp. 69–75.
51. Raschka, S. *Python Machine Learning*, 1st ed.; Packt Publishing Ltd.: Birmingham, UK, 2015; p. 425. ISBN 978-1-78355-513-0.
52. Buro, M. Improving heuristic mini-max search by supervised learning. *Artif. Intell.* **2002**, *134*, 85–99. [CrossRef]
53. Mendes, A.; Togelius, J.; Nealen, A. Hyper-heuristic General Video Game Playing. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Santorini, Greece, 20–23 September 2016; pp. 102–109. [CrossRef]
54. Laskey, M.; Staszak, S.; Hsieh, W.Y.S.; Mahler, J.; Pokorny, F.T.; Dragan, A.D.; Goldberg, K. SHIV: Reducing Supervisor Burden in DAgger using Support Vectors for Efficient Learning from Demonstrations in High Dimensional State Spaces. In Proceedings of the IEEE International Conference on Robotics and Automation, Stockholm, Sweden, 16–21 May 2016; pp. 462–469. [CrossRef]
55. Young, J.; Hawes, N. Learning Micro-Management Skills in RTS Games by Imitating Experts. In Proceedings of the 10th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Raleigh, NC, USA, 3–7 October 2014; AAAI Press; pp. 195–201.
56. Park, H.; Cho, H.C.; Lee, K.Y.; Kim, K.J. Prediction of Early Stage Opponents Strategy for StarCraft AI using Scouting and Machine Learning. In Proceedings of the WASA 2012: Workshop at SIGGRAPH Asia, Singapore, 26–27 November 2012; pp. 7–12. [CrossRef]
57. Asayama, K.; Moriyama, K.; Fukui, K.I.; Numao, M. Prediction as Faster Perception in a Real-Time Fighting Video Game. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Tainan, Taiwan, 31 August–2 September 2015; pp. 517–522. [CrossRef]
58. Weber, B.G.; John, M.; Mateas, M.; Jhala, A. Modeling player retention in Madden NFL 11. In Proceedings of the 23rd Innovative Applications of Artificial Intelligence Conference, San Francisco, CA, USA, 9–11August 2011; pp. 1701–1706.
59. Weber, B.G.; Mateas, M.; Jhala, A. Using Data Mining to Model Player Experience. In Proceedings of the 2nd International Workshop on Evaluating Player Experience in Games, Bordeaux, France, 28 June–1 July 2011.
60. Zocca, V.; Spacagna, G.; Slater, D.; Roelants, P. *Python Deep Learning*; Packt Publishing Ltd.: Birmingham, UK, 2017; p. 383. ISBN 978-1-78646-445-3.
61. Tesauro, G. Neurogammon: A Neural-Network Backgammon Program. In Proceedings of the International Joint Conference on Neural Networks, San Diego, CA, USA, 17–21 June 1990; pp. 33–39. [CrossRef]
62. Tesauro, G. Connectionist Learning of Expert Preferences by Comparison Training. In *Advances in Neural Information Processing Systems 1 (NIPS 1988)*; Morgan Kaufmann: Burlington, MA, USA, 1989; pp. 99–106. ISBN 1-55860-015-9.
63. Griffith, N.J.L.; Lynch, M. NeuroDraughts: The role of representation, search, training regime and architecture in a TD draughts player. In Proceedings of the 8th Ireland Conference on Artificial Intelligence, Ulster, UK, 10–13 September 1997; pp. 64–72.
64. Thrun, S. Learning To Play the Game of Chess. In *Advances in Neural Information Processing Systems 7 (NIPS 1994)*; The MIT Press: Cambridge, MA, USA, 1995; pp. 1069–1076. ISBN 97802622010491167.
65. Gemine, Q.; Safadi, F.; Fonteneau, R.; Ernst, D. Imitative Learning for Real-Time Strategy Games. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Granada, Spain, 11–14 September 2012; pp. 424–429. [CrossRef]
66. Oh, J.; Guo, X.; Lee, H.; Lewis, R.; Singh, S. Action-Conditional Video Prediction using Deep Networks in Atari Games. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*; Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; Volume 2, pp. 2863–2871.

67. Luo, Z.; Guzdial, M.; Liao, N.; Riedl, M. Player Experience Extraction from Gameplay Video. In Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Edmonton, Canada, 13–17 November 2018; pp. 52–58.
68. Gaspareto, O.B.; Barone, D.A.C.; Schneider, A.M. Neural Networks Applied to Speed Cheating Detection in Online Computer Games. In Proceedings of the 4th International Conference on Natural Computation, Jinan, China, 18–20 October 2008; pp. 526–529. [CrossRef]
69. Mori, K.; Ito, S.; Harada, T.; Thawonmas, R.; Kim, K.J. Feature Extraction of Gameplays for Similarity Calculation in Gameplay Recommendation. In Proceedings of the IEEE 10th International Workshop on Computational Intelligence and Applications, Hiroshima, Japan, 11–12 November 2017; pp. 171–176. [CrossRef]
70. Manchin, A.; Abbasnejad, E.; van den Hengel, A. Reinforcement Learning with Attention that Works: A Self-Supervised Approach. In *Neural Information Processing. ICONIP 2019. Communications in Computer and Information Science*; Gedeon, T., Wong, K., Lee, M., Eds.; Springer: Cham, Switzerland, 2019; Volume 1143, pp. 223–230. ISBN 978-3-030-36802-9.
71. Racanière, S.; Weber, T.; Reichert, D.P.; Buesing, L.; Guez, A.; Rezende, D.; Badia, A.P.; Vinyals, O.; Heess, N.; Li, Y.; et al. Imagination-Augmented Agents for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*; Curran Associates Inc.: Red Hook, NY, USA, 2017; pp. 5694–5705.
72. Pathak, D.; Agrawal, P.; Efros, A.A.; Darrell, T. Curiosity-driven Exploration by Self-supervised Prediction. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 2778–2787.
73. Ha, D.; Schmidhuber, J. World Models. *Zenodo* **2018**. [CrossRef]
74. Peng, P.; Wen, Y.; Yang, Y.; Yuan, Q.; Tang, Z.; Long, H.; Wang, J. Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games. *arXiv* **2017**, arXiv:1703.10069.
75. Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computing*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2015; p. 287. ISBN 978-3-662-44874-8.
76. Miikkulainen, R.; Bryant, B.D.; Cornelius, R.; Karpov, I.V.; Stanley, K.O.; Yong, C.H. Computational Intelligence in Games. In *Computational Intelligence: Principles and Practice*; Yen, G.Y., Fogel, D.B., Eds.; IEEE: Piscataway, NJ, USA, 2006; ISBN 9780978713508.
77. Risi, S.; Togelius, J. Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Trans. Comput. Intell. AI Games* **2017**, *9*, 25–41. [CrossRef]
78. Richards, N.; Moriarty, D.E.; Miikkulainen, R. Evolving Neural Networks to Play Go. *Appl. Intell.* **1998**, *8*, 85–96. [CrossRef]
79. Fogel, D.B.; Hays, T.J.; Hahn, S.L.; Quon, J. A Self-Learning Evolutionary Chess Program. *Proc. IEEE* **2004**, *92*, 1947–1954. [CrossRef]
80. Fogel, D.B.; Chellapilla, K. Verifying Anaconda's expert rating by competing against Chinook: Experiments in co-evolving a neural checkers player. *Neurocomputing* **2002**, *42*, 69–86. [CrossRef]
81. Neto, H.C.; Julia, R.M.S.; Caexeta, G.S.; Barcelos, A.R.A. LS-VisionDraughts: Improving the performance of an agent for checkers by integrating computational intelligence, reinforcement learning and a powerful search method. *Appl. Intell.* **2014**, *41*, 525–550. [CrossRef]
82. Ortega, J.; Shaker, N.; Togelius, J.; Yannakakis, G.N. Imitating human playing styles in Super Mario Bros. *Entertain. Comput.* **2013**, *4*, 93–104. [CrossRef]
83. Salcedo-Sanz, S.; Matías-Román, J.M.; Jiménez-Fernández, S.; Portilla-Figueras, A.; Cuadra, L. An evolutionary-based hyper-heuristic approach for the Jawbreaker puzzle. *Appl. Intell.* **2014**, *40*, 404–414. [CrossRef]
84. Pedersen, C.; Togelius, J.; Yannakakis, G.N. Modeling Player Experience in Super Mario Bros. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Milano, Italy, 7–10 September 2009; pp. 132–139. [CrossRef]
85. Togelius, J.; Schmidhuber, J. An Experiment in Automatic Game Design. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Perth, Australia, 15–18 December 2008; pp. 111–118. [CrossRef]

86. Hom, V.; Marks, J. Automatic Design of Balanced Board Games. In Proceedings of the 3rd Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, California, 6–8 June 2007; pp. 25–30.
87. Mourato, F.; Dos Santos, M.P.; Birra, F. Automatic level generation for platform videogames using Genetic Algorithms. In Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology, Lisbon, Portugal, 8–11 November 2011; pp. 54–61. [CrossRef]
88. Hastings, E.; Guha, R.; Stanley, K.O. NEAT Particles: Design, Representation, and Animation of Particle System Effects. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Honolulu, HI, USA, 1–5 April 2007; pp. 154–160. [CrossRef]
89. Hastings, E.J.; Guha, R.K.; Stanley, K.O. Automatic Content Generation in the Galactic Arms Race Video Game. *IEEE Trans. Comput. Intell. AI Games* **2009**, *1*, 245–263. [CrossRef]
90. Togelius, J.; De Nardi, R.; Lucas, S.M. Towards automatic personalised content creation for racing games. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Honolulu, HI, USA, 1–5 April 2007; pp. 252–259. [CrossRef]
91. Jim, K.C.; Giles, C.L. Talking helps: Evolving communicating agents for the predator-prey pursuit problem. *Artif. Life* **2000**, *6*, 237–254. [CrossRef] [PubMed]
92. Bishop, J.; Miikkulainen, R. Evolutionary Feature Evaluation for Online Reinforcement Learning. In Proceedings of the Conference on Computatonal Intelligence and Games, Niagara Falls, ON, Canada, 11–13 August 2013; pp. 267–274. [CrossRef]
93. Davison, T.; Denzinger, J. The Huddle: Combining AI Techniques to Coordinate a Player's Game Characters. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Granada, Spain, 11–14 September 2012; pp. 203–210. [CrossRef]
94. Stanley, K.O.; Bryant, B.D.; Miikkulainen, R. Real-Time Neuroevolution in the NERO Video Game. *IEEE Trans. Evol. Comput.* **2005**, *9*, 653–668. [CrossRef]
95. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2018; ISBN 9780262039246.
96. Watkins, C.J.C.H.; Dayan, P. Q-Learning. *Mach. Learn* **1992**, *8*, 279–292. [CrossRef]
97. Watkins, C.J.C.H. *Learning from Delayed Rewards*; King's College: Cambridge, UK, 1989.
98. Tesauro, G. Practical Issues in Temporal Difference Learning. *Mach. Learn* **1992**, *8*, 257–277. [CrossRef]
99. Samuel, A.L. Some Studies in Machine Learning Using the Game of Checkers. *IBM J. Res. Dev.* **1959**, *3*, 210–229. [CrossRef]
100. Huang, H.H.; Wang, T. Learning Overtaking and Blocking Skills in Simulated Car Racing. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Tainan, Taiwan, 31 August–2 September 2015; pp. 439–445. [CrossRef]
101. Abbeel, P.; Ng, A.Y. Apprenticeship Learning via Inverse Reinforcement Learning. In Proceedings of the 21st International Conference on Machine Learning, Banff, AB, Canada; 2004; pp. 1–8. Available online: https://ai.stanford.edu/~{}ang/papers/icml04-apprentice.pdf (accessed on 29 May 2020). [CrossRef]
102. Hester, T.; Vecerik, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Horgan, D.; Quan, J.; Sendonaris, A.; Osband, I.; et al. Deep Q-learning from Demonstrations. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 3223–3230.
103. Tastan, B.; Sukthankar, G. Learning Policies for First Person Shooter Games Using Inverse Reinforcement Learning. In Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 10–14 October 2011; pp. 85–90.
104. Knox, W.B.; Stone, P. Reinforcement Learning from Simultaneous Human and MDP Reward. In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, Valencia, Spain, 4–8 June 2012; International Foundation for Autonomous Agents and Multiagent Systems. Volume 1, pp. 475–482.
105. Knox, W.B.; Stone, P. Interactively Shaping Agents via Human Reinforcement The TAMER Framework. In Proceedings of the 5th International Conference on Knowledge Capture, Redondo Beach, CA, USA, 1–4 September 2009; pp. 9–16. [CrossRef]
106. Shaker, N. Intrinsically Motivated Reinforcement Learning: A Promising Framework for Procedural Content Generation. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Santorini, Greece, 20–23 September 2016; pp. 1–8. [CrossRef]

107. Usunier, N.; Synnaeve, G.; Lin, Z.; Chintala, S. Episodic Exploration for Deep Deterministic Policies for StarCraft Micromanagement. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
108. Wender, S.; Watson, I. Applying Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game StarCraft:Broodwar. In Proceedings of the Conference on Computational Intelligence and Games, Granada, Spain, 11–14 September 2012; pp. 402–408. [CrossRef]
109. Bellemare, M.G.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; Munos, R. Unifying Count-Based Exploration and Intrinsic Motivation. In Proceedings of the Advances in Neural Information Processing Systems 29 (NIPS 2016), Barcelona, Spain, 5–10 December 2016; pp. 1471–1479.
110. Sukhbaatar, S.; Lin, Z.; Kostrikov, I.; Synnaeve, G.; Szlam, A.; Fergus, R. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In Proceedings of the 6th International Conference on Learning Representations (ICLR 2018), Vancouver, Canada, 30 April—3 May 2018.
111. Kulkarni, T.D.; Narasimhan, K.R.; Saeedi, A.; Tenenbaum, J.B. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In Proceedings of the Advances in Neural Information Processing Systems 29 (NIPS 2016), Barcelona, Spain, 5–10 December 2016; pp. 3675–3683.
112. Burda, Y.; Edwards, H.; Pathak, D.; Storkey, A.; Darrell, T.; Efros, A.A. Large-Scale Study of Curiosity-Driven Learning. In Proceedings of the 7th International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6–9 May 2019.
113. Wu, Y.; Tian, Y. Training Agent for First-Person Shooter Game with Actor-Critic Curriculum Learning. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
114. Hamrick, J.B.; Ballard, A.J.; Pascanu, R.; Vinyals, O.; Heess, N.; Battaglia, P.W. Metacontrol for Adaptive imagination-Based Optimization. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
115. Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; Mordatch, I. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 4–9 Devember2017; pp. 6379–6390.
116. Baker, B.; Kanitscheider, I.; Markov, T.; Wu, Y.; Powell, G.; McGrew, B.; Mordatch, I. Emergent Tool Use From Multi-Agent Autocurricula. In Proceedings of the 8th International Conference on Learning Representations, Addis Ababa, Ethiopia, 26 April–1 May 2020.
117. Bansal, T.; Pachocki, J.; Sidor, S.; Sutskever, I.; Mordatch, I. Emergent Complexity via Multi-Agent Competition. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
118. Jaderberg, M.; Czarnecki, W.M.; Dunning, I.; Marris, L.; Lever, G.; Castañeda, A.G.; Beattie, C.; Rabinowitz, N.C.; Morcos, A.S.; Ruderman, A.; et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science* **2019**, *364*, 859–865. [CrossRef]
119. Koller, D.; Friedman, N. *Probabilistic Graphical Models Principles and Techniques*; The MIT Press: Cambridge, MA, USA, 2009; p. 1233. ISBN 978-0-262-01319-2.
120. Albrecht, D.W.; Zukerman, I.; Nicholson, A.E. Bayesian Models for Keyhole Plan Recognition in an Adventure Game. *User Model. User-adapt. Interact.* **1998**, *8*, 5–47. [CrossRef]
121. Hostetler, J.; Dereszynski, E.; Dietterich, T.; Fern, A. Inferring Strategies from Limited Reconnaissance in Real-time Strategy Games. In Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, 14–18 August 2012; pp. 367–376.
122. Synnaeve, G.; Bessière, P. A Bayesian Model for Plan Recognition in RTS Games Applied to StarCraft. In Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 10–14 October 2011; pp. 79–84.
123. Leece, M.; Jhala, A. Opponent State Modeling In RTS games with limited information using Markov Random Fields. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 139–145. [CrossRef]
124. Parra, R.; Garrido, L. Bayesian Networks for Micromanagement Decision Imitation in the RTS Game Starcraft. In Proceedings of the 11th Mexican international conference on Advances in Computational Intelligence, San Luis Potosí, Mexico, 27 October–4 November 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 433–443. [CrossRef]

125. Synnaeve, G.; Bessière, P. A Bayesian Model for RTS Units Control applied to StarCraft. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Seoul, Korea, 31 August–3 September 2011; pp. 190–196. [CrossRef]
126. Langley, P.; Simon, H.A. Applications of Machine Learning and Rule Induction. Commun. *ACM* **1995**, *38*, 55–64. [CrossRef]
127. Agnar, A.; Plaza, E. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Commun.* **1994**, *7*, 39–59. [CrossRef]
128. Aha, D.W.; Molineaux, M.; Ponsen, M. Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game. In Proceedings of the International Conference on Case-Based Reasoning, Chicago, IL, USA, 23–26 August 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 5–20. [CrossRef]
129. Weber, B.G.; Mateas, M. Case-Based Reasoning for Build Order in Real-Time Strategy Games. In Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 14–16 October 2009; pp. 106–111.
130. Szczepanski, T.; Aamodt, A. Case-based Reasoning for Improved Micromanagement in Real-Time Strategy Games. In Proceedings of the Workshop on Case-Based Reasoning for Computer Games, 8th International Conference on Case-Based Reasoning, Seattle, WA, USA, 20–23 July 2009; pp. 139–148.
131. Hsieh, J.L.; Sun, C.T. Building a Player Strategy Model by Analyzing Replays of Real-Time Strategy Games. In Proceedings of the IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–8 June 2008; pp. 3106–3111. [CrossRef]
132. Spronck, P.; Sprinkhuizen-Kuyper, I.; Postma, E. Online Adaptation of Game Opponent AI with Dynamic Scripting. *Int. J. Intell. Games Simul.* **2004**, *3*, 45–53.
133. Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; Postma, E. Adaptive game AI with dynamic scripting. *Mach. Learn* **2006**, *63*, 217–248. [CrossRef]
134. Ponsen, M.; Spronck, P. Improving Adaptive Game AI with Evolutionary Learning. In Proceedings of the 5th Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education, Wolverhampton, UK, 8–10 November 2004; pp. 389–396.
135. Bellemare, M.G.; Veness, J. The Arcade Learning Environment: An Evaluation Platform for General Agents. *J. Artif. Intell. Res.* **2013**, *47*, 253–279. [CrossRef]
136. Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; Jaskowski, W. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Santorini, Greece, 20–23 September 2016; pp. 373–380. [CrossRef]
137. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A physics engine for model-based control. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, Portugal, 7–12 Ocober 2012; pp. 5026–5033. [CrossRef]
138. Juliani, A.; Berges, V.-P.; Vckay, E.; Gao, Y.; Henry, H.; Mattar, M.; Lange, D. Unity: A General Platform for Intelligent Agents. *arXiv* **2018**, arXiv:1809.02627.
139. Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; Sutskever, I. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv* **2017**, arXiv:1703.03864.
140. Yin, H.; Luo, L.; Cai, W.; Ong, Y.S.; Zhong, J. A Data-driven Approach for Online Adaptation of Game Difficult. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Tainan, Taiwan, 31 August–2 September 2015; pp. 146–153. [CrossRef]
141. Rabin, S. *Game AI Pro 2: Collected Wisdom of Game AI Professionals*; CRC Press: Boca Raton, FL, USA, 2015; ISBN 9781482254808.
142. Rabin, S. *Game AI Pro: Collected Wisdom of Game AI Professionals*; CRC Press: Boca Raton, FL, USA, 2014; ISBN 9781466565975.
143. Rabin, S. *AI Game Programming Wisdom*; Charles River Media: Needham Heights, MA, USA, 2002; ISBN 1584500778.
144. Peng, J.; Williams, R.J. E?cient Learning and Planning Within the Dyna Framework. *Adapt. Behav.* **1993**, *1*, 437–454. [CrossRef]
145. Wang, Y.; Gelly, S. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Honolulu, HI, USA, 1–5 April 2007; pp. 175–182. [CrossRef]

146. Coulom, R. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In Proceedings of the 5th International Conference on Computer and Games, Turin, Italy, 29–31 May 2006; pp. 72–83. [CrossRef]
147. Churchill, D.; Saffidine, A.; Buro, M. Fast Heuristic Search for RTS Game Combat Scenarios. In Proceedings of the 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 8–12 October 2012; pp. 112–117.
148. Silver, D. Cooperative Pathfinding. In Proceedings of the 1st AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Marina del Rey, CA, USA, 1–3 June 2005; pp. 117–122.
149. Harabor, D.; Botea, A. Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Perth, Australia, 15–18 December 2008; pp. 258–265. [CrossRef]
150. Maggiore, G.; Santos, C.; Dini, D.; Peters, F.; Bouwknegt, H.; Spronck, P. LGOAP: Adaptive layered planning for real-time videogames. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Niagara Falls, ON, Canada, 11–13 August 2013; pp. 283–290. [CrossRef]
151. Drake, J.; Safonova, A.; Likhachev, M. Towards Adaptability of Demonstration-Based Training of NPC Behavior. In Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Snowbird, UT, USA, 5–9 October 2017; pp. 179–185.
152. Tremblay, J.; Borodovski, A.; Verbrugge, C. I Can Jump! Exploring Search Algorithms for Simulating Platformer Players. In Proceedings of the 10th Artificial Intelligence and Interactive Digital Entertainment Conference, Raleigh, NC, USA, 3–7 October 2014; pp. 58–64.
153. Togelius, J.; Karakovskiy, S.; Baumgarten, R. The 2009 Mario AI Competition. In Proceedings of the IEEE Congress on Evolutionary Computation, Barcelona, Spain, 18–23 July 2010; pp. 1–8. [CrossRef]
154. Ghallab, M.; Nau, D.; Traverso, P. *Automated Planning Theory and Practice*; Morgan Kaufmann Publishers: San Francisco, CA, USA, 2004; ISBN 1-55860-856-7.
155. Menif, A.; Jacopin, É.; Cazenave, T. SHPE: HTN Planning for Video Games. In *Computer Games. CGW 2014. Communications in Computer and Information Science*; Cazenave, T., Winands, M.H., Björnsson, Y., Eds.; Springer: Cham, Switzerland, 2014; Volume 504, pp. 119–132. ISBN 978-3-319-14923-3.
156. Nguyen, C.; Reifsnyder, N.; Gopalakrishnan, S.; Munoz-Avila, H. Automated Learning of Hierarchical Task Networks for Controlling Minecraft Agents. In Proceedings of the IEEE Conference on Computational Intelligence and Games, New York, NY, USA, 22–25 August 2017; pp. 226–231. [CrossRef]
157. Hoang, H.; Lee-Urban, S.; Muñoz-Avila, H. Hierarchical Plan Representations for Encoding Strategic Game AI. In Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, Marina del Rey, CA, USA, 1–3 June 2005; pp. 63–68.
158. Gorniak, P.; Davis, I. SquadSmart Hierarchical Planning and Coordinated Plan Execution for Squads of Characters. In Proceedings of the 3rd Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 6–8 June 2007; pp. 14–19.
159. Neufeld, X.; Mostaghim, S.; Perez-Liebana, D. HTN Fighter: Planning in a Highly-Dynamic Game. In Proceedings of the 9th Conference on Computer Science and Electronic Engineering, Colchester, UK, 27–29 September 2017; pp. 189–194. [CrossRef]
160. Kelly, J.P.; Botea, A.; Koenig, S. Offline Planning with Hierarchical Task Networks in Video Games. In Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 22–24 October 2008; pp. 60–65.
161. Guilherme Da Silva, F.A.; Feijó, B. Personality Traits in Plots with Nondeterministic Planning for Interactive Storytelling. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Tainan, Taiwan, 31 August–2 September 2015; pp. 83–90. [CrossRef]
162. Neufeld, X.; Mostaghim, S.; Perez-Liebana, D. A Hybrid Planning and Execution Approach Through HTN and MCTS. In Proceedings of the 3rd Workshop on Integrated Planning, Acting, and Execution, Berkeley, CA, USA, 12 July 2019; pp. 37–45.
163. Kocsis, L.; Szepesvári, C. Bandit based Monte-Carlo Planning. In *Machine Learning: ECML 2006. ECML 2006. Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4212, pp. 282–293. ISBN 978-3-540-45375-8.
164. 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* **2012**, *4*, 1–43. [CrossRef]

165. Dann, M.; Zambettau, F.; Thangarajah, J. An Improved Approach to Reinforcement Learning in Computer Go. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Tainan, Taiwan, 31 August–2 September 2015; pp. 169–176. [CrossRef]
166. Barrett, S.; Stone, P.; Kraus, S. Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain. In Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems, Taipei, Taiwan, 2–6 May 2011; Volume 2, pp. 567–574, ISBN 0-9826571-5-3.
167. Williams, P.R.; Walton-Rivers, J.; Perez-Liebana, D.; Lucas, S.M. Monte Carlo Tree Search Applied to Co-operative Problems. In Proceedings of the 7th Computer Science and Electronic Engineering Conference, Colchester, UK, 24–25 Sepember 2015; pp. 219–224. [CrossRef]
168. Demediuk, S.; Tamassia, M.; Raffe, W.L.; Zambetta, F.; Li, X.; Mueller, F. Monte Carlo Tree Search Based Algorithms for Dynamic Difficulty Adjustment. In Proceedings of the IEEE Conference on Computational Intelligence and Games, New York, NY, USA, 22–25 August 2017; pp. 53–59. [CrossRef]
169. Devlin, S.; Anspoka, A.; Sephton, N.; Cowling, P.I.; Rollason, J. Combining Gameplay Data With Monte Carlo Tree Search To Emulate Human Play. In Proceedings of the 12th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Burlingame, CA, USA, 8–12 October 2016; pp. 16–22.
170. Whitehouse, D.; Cowling, P.I.; Powley, E.J.; Rollason, J. Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game. In Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Boston, MA, USA, 14–15 October 2013; pp. 100–106.
171. Bowen, N.; Todd, J.; Sukthankar, G. Adjutant Bot: An Evaluation of Unit Micromanagement Tactics. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Niagara Falls, ON, Canada, 11–13 August 2013; pp. 336–343. [CrossRef]
172. Justesen, N.; Tillman, B.; Togelius, J.; Risi, S. Script- and Cluster-based UCT for StarCraft. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 317–324. [CrossRef]
173. Balla, R.; Fern, A. UCT for Tactical Assault Planning in Real-Time Strategy Games. In Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, CA, USA, 11–17 July 2009; pp. 40–45.
174. Ishii, R.; Ito, S.; Ishihara, M.; Harada, T.; Thawonmas, R. Monte-Carlo Tree Search Implementation of Fighting Game AIs Having Personas. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Maastricht, Netherlands, 14–17 August 2018; pp. 54–61. [CrossRef]
175. Sarratt, T.; Pynadath, D.V.; Jhala, A. Converging to a Player Model In Monte-Carlo Tree Search. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 169–175. [CrossRef]
176. Sanselone, M.; Sanchez, S.; Sanza, C.; Panzoli, D.; Duthen, Y. Constrained control of non-playing characters using Monte Carlo Tree Search. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 208–215. [CrossRef]
177. Sironi, C.F.; Liu, J.; Perez-Liebana, D.; Gaina, R.D.; Bravi, I.; Lucas, S.M.; Winands, M.H.M. Self-Adaptive MCTS for General Video Game Playing. In *Applications of Evolutionary Computation. EvoApplications 2018. Lecture Notes in Computer Science*; Sim, K., Kaufmann, P., Eds.; Springer: Cham, Switzerland, 2018; Volume 10784, pp. 358–375. ISBN 978–3-319-77538-8.
178. Sironi, C.F.; Winands, M.H.M. Analysis of Self-Adaptive Monte Carlo Tree Search in General Video Game Playing. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Maastricht, The Netherlands, 14–17 August 2018; pp. 397–400. [CrossRef]
179. Sironi, C.F.; Liu, J.; Winands, M.H.M. Self-Adaptive Monte-Carlo Tree Search in General Game Playing. *IEEE Trans. Games (Early Access)* **2018**. [CrossRef]
180. Fikes, R.E.; Nilsson, N.J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.* **1971**, *2*, 189–208. [CrossRef]
181. Pednault, E.P.D. ADL and the State-Transition Model of Action. *J. Log. Comput.* **1994**, *4*, 467–512. [CrossRef]
182. Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; et al. PDDL—The Planning Domain Definition Language. *AIPS Plan. Comm.* **1998**, *3*, 14.
183. Orkin, J. Three States and a Plan: The A.I. of F.E.A.R. In Proceedings of the Game Developers Conference—GDC 2006, San Jose, CA, USA, 20–24 March 2006.

184. Kovarsky, A.; Buro, M. A First Look at Build-Order Optimization in Real-Time Strategy Games. In Proceedings of the 7th International Conference on Intelligent Games and Simulation, Braunschweig, Germany, 29 November–1 December 2006; pp. 60–64.
185. Mateas, M.; Stern, A. A Behavior Language for Story-Based Believable Agents. *IEEE Intell. Syst.* **2002**, *17*, 39–47. [CrossRef]
186. Weber, B.G.; Mawhorter, P.; Mateas, M.; Jhala, A. Reactive Planning Idioms for Multi-Scale Game AI. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Dublin, Ireland, 18–21 August 2010; pp. 115–122. [CrossRef]
187. Branquinho, A.A.B.; Lopes, C.R. Planning for Resource Production in Real-Time Strategy Games Based on Partial Order Planning, Search and Learning. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Istanbul, Turkey, 10–13 October 2010; pp. 4205–4211. [CrossRef]
188. Shirvani, A.; Farrell, R.; Ware, S.G. Combining Intentionality and Belief: Revisiting Believable Character Plans. In Proceedings of the 14th Artificial Intelligence and Interactive Digital Entertainment Conference, Edmonton, Canada, 13–17 November 2018; pp. 222–228.
189. Van Lent, M.; Riedl, M.O.; Carpenter, P.; Mcalinden, R.; Brobst, P. Increasing Replayability with Deliberative and Reactive Planning. In Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, Marina del Rey, CA, USA, 1–3 June 2005; pp. 135–140.
190. Weber, B.G.; Mateas, M.; Jhala, A. Applying Goal-Driven Autonomy to StarCraft. In Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Palo Alto, CA, USA, 11–13 October 2010; pp. 101–106.
191. Labranche, S.; Sola, N.; Callies, S.; Beaudry, E. Using Partial Satisfaction Planning to Automatically Select NPCs' Goals and Generate Plans In a Simulation Game. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 240–247. [CrossRef]
192. Abelha, P.; Gottin, V.; Ciarlini, A.; Araujo, E.; Furtado, A.; Feijo, B.; Silva, F.; Pozzer, C. A Nondeterministic Temporal Planning Model for Generating Narratives with Continuous Change in Interactive Storytelling. In Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Boston, MA, USA, 14–18 October 2013; pp. 107–113.
193. Shirvani, A.; Ware, S.G.; Farrell, R. A Possible Worlds Model of Belief for State-Space Narrative Planning. In Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Little Cottonwood Canyon, UT, USA, 5–9 October 2017; pp. 101–107.
194. Ware, S.G.; Young, R.M. Glaive: A State-Space Narrative Planner Supporting Intentionality and Conflict. In Proceedings of the 10th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Raleigh, NC, USA, 3–7 October 2014; pp. 80–86.
195. Li, B.; Riedl, M.O. An Offline Planning Approach to Game Plotline Adaptation. In Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Palo Alto, CA, USA, 11–13 October 2010; pp. 45–50.
196. LaValle, S.M. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. 1998. Available online: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=BDCAC5E777FC7B257783418EE8F274C4?doi=10.1.1.35.1853&rep=rep1&type=pdf (accessed on 29 May 2020).
197. Shepherd, J.J.; Tang, J.; O'Kane, J. Path-Finding with Motion Constraints in Real Time Strategies. In Proceedings of the Annual International Conferences on Computer Games, Multimedia and Allied Technology, Amara, Singapore, 11–12 May 2009; pp. 83–90. [CrossRef]
198. Xu, S. Improving Companion Ai in Small-Scale Attrition Games. Master's Thesis, McGill University, Montreal, Canada, October 2015.
199. Soares, R.; Leal, F.; Prada, R.; Melo, F. Rapidly-Exploring Random Tree approach for Geometry Friends. In Proceedings of the 1st International Joint Conference of DiGRA and FDG, Dundee, UK, 1–6 August 2016.
200. Bauer, A.; Popović, Z. RRT-Based Game Level Analysis, Visualization, and Visual Refinement. In Proceedings of the 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 8–12 October 2012; pp. 8–13.
201. Bauer, A.; Cooper, S.; Popović, Z. Automated Redesign of Local Playspace Properties. In Proceedings of the 8th International Conference on the Foundations of Digital Games, Chania, Greece, 14–17 May 2013; pp. 190–197.

202. Tremblay, J.; Torres, P.A.; Rikovitch, N.; Verbrugge, C. An Exploration Tool for Predicting Stealthy Behaviour. In Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AAAI Technical Report WS-13-20. Boston, MA, USA, 14–18 October 2013; 2013; pp. 34–40.
203. Osborn, J.C.; Lambrigger, B.; Mateas, M. HyPED: Modeling and Analyzing Action Games as Hybrid Systems. In Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Snowbird, UT, USA, 5–9 October 2017; pp. 87–93.
204. Spalazzi, L. A Survey on Case-Based Planning. *Artif. Intell. Rev.* **2001**, *16*, 3–36. [CrossRef]
205. Ontañón, S.; Mishra, K.; Sugandh, N.; Ram, A. On-Line Case-Based Planning. *Comput. Intell.* **2010**, *26*, 84–119. [CrossRef]
206. Coman, A.; Muñoz-Avila, H. Plan-Based Character Diversity. In Proceedings of the 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, MA, USA, 8–12 October 2012; pp. 118–123.
207. Muñoz-Avila, H.; Aha, D.W. On the Role of Explanation for Hierarchical Case- Based Planning in Real-Time Strategy Games. In *Case-Based Reasoning Research and Development. ICCBR 2007. Lecture Notes in Computer Science*; Weber, R.O., Ed.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4626, pp. 164–178. ISBN 978-3-540-74141-1.
208. Ontañón, S.; Mishra, K.; Sugandh, N.; Ram, A. Case-Based Planning and Execution for Real-Time Strategy Games. In *Case-Based Reasoning Research and Development. ICCBR 2007. Lecture Notes in Computer Science*; Weber, R.O., Ed.; Springer: Berlin/Heidelberg, Germany; Belfast, UK, 2007; Volume 4626, pp. 164–178. ISBN 978-3-540-74141-1.
209. Ontañón, S.; Bonnette, K.; Mahindrakar, P.; Gómez-Martín, M.A.; Long, K.; Radhakrishnan, J.; Shah, R.; Ram, A. Learning from Human Demonstrations for Real-Time Case-Based Planning. In Proceedings of the IJCAI-09 Workshop on Learning Structural Knowledge From Observations, Pasadena, CA, USA, 12 July 2019.
210. Cheng, D.C.; Thawonmas, R. Case-based plan recognition for real-time strategy games. In Proceedings of the 5th Game-On International Conference, Reading, UK, 8–10 November 2004; pp. 36–40.
211. Colledanchise, M.; Ögren, P. *Behavior Trees in Robotics and AI: An Introduction*, 1st ed.; CRC Press: Boca Raton, FL, USA, 2018; ISBN 9781138593732.
212. Ocio, S. Adapting AI Behaviors To Players in Driver San Francisco Hinted-Execution Behavior Trees. In Proceedings of the 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, MA, USA, 8–12 October 2012; pp. 51–56. [CrossRef]
213. Johansson, A.; Dell'Acqua, P. Emotional Behavior Trees. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Granada, Spain, 11–14 September 2012; pp. 355–362. [CrossRef]
214. Tremblay, J.; Verbrugge, C. Adaptive Companions in FPS Games. In Proceedings of the 8th International Conference on the Foundations of Digital Games, Chania, Greece, 14–17 May 2013; pp. 229–236.
215. Zhang, Q.; Yao, J.; Yin, Q.; Zha, Y. Learning Behavior Trees for Autonomous Agents with Hybrid Constraints Evolution. *Appl. Sci.* **2018**, *8*, 77. [CrossRef]
216. Colledanchise, M.; Parasuraman, R.; Ogren, P. Learning of Behavior Trees for Autonomous Agents. *IEEE Trans. Games* **2019**, *11*, 183–189. [CrossRef]
217. Dey, R.; Child, C. QL-BT: Enhancing Behaviour Tree Design and Implementation with Q-Learning. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Niagara Falls, ON, Canada, 11–13 August 2013; pp. 275–282. [CrossRef]
218. Kadlec, R.; Tóth, C.; Černý, M.; Barták, R.; Brom, C. Planning Is the Game: Action Planning as a Design Tool and Game Mechanism. In Proceedings of the 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 8–12 October 2012; pp. 160–166.
219. Horswill, I.D. Postmortem: MKULTRA, An Experimental AI-Based Game. In Proceedings of the 14th Artificial Intelligence and Interactive Digital Entertainment Conference, Edmonton, AB, Canada, 13–17 November 2018; pp. 45–51.
220. Robison, E. A Sandbox for Modeling Social AI. In Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Workshop on Experimental AI in Games WS-17-19, Snowbird, UT, USA, 5–9 October 2017; pp. 107–110.
221. Cash, S.P.; Young, R.M. Bowyer: A Planning Tool for Bridging the Gap between Declarative and Procedural Domains. In Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 14–16 October 2009; pp. 14–19.

222. Ghallab, M.; Nau, D.; Traverso, P. The actor's view of automated planning and acting: A position paper. *Artif. Intell.* **2014**, *208*, 1–17. [CrossRef]
223. Alcázar, V.; Guzmán, C.; Prior, D.; Borrajo, D.; Castillo, L.; Onaindia, E. PELEA: Planning, Learning and Execution Architecture. In Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group, Brescia, Italy, 1–2 December 2010; pp. 17–24.
224. Guzmán, C.; Alcázar, V.; Prior, D.; Onaindia, E.; Borrajo, D.; Fdez-Olivares, J.; Quintero, E. PELEA: A Domain-Independent Architecture for Planning, Execution and Learning. In Proceedings of the 6th Scheduling and Planning Applications woRKshop, São Paulo, Brazil, 26 June 2012; pp. 38–45.
225. Yang, F.; Lyu, D.; Liu, B.; Gustafson, S. PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making. In Proceedings of the 27th International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 13–19 July 2018; pp. 4860–4866.
226. Neufeld, X.; Mostaghim, S.; Sancho-Pradel, D.L.; Brand, S. Building a Planner: A Survey of Planning Systems Used in Commercial Video Games. *IEEE Trans. Games* **2019**, *11*, 91–108. [CrossRef]
227. Grzes, M.; Kudenko, D. Plan-based Reward Shaping for Reinforcement Learning. In Proceedings of the 4th IEEE International Conference on Intelligent Systems, Varna, Bulgaria, 6–8 September 2008; pp. 22–29. [CrossRef]
228. Guo, X.; Singh, S.; Lee, H.; Lewis, R.; Wang, X. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In Proceedings of the Advances in Neural Information Processing Systems 27 (NIPS 2014), Montreal, QC, Canada, 8–13 December 2014; pp. 3338–3346.
229. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. In Proceedings of the NIPS Deep Learning Workshop, Lake Tahoe, CA, USA, 9 December 2013.
230. Tesauro, G. Temporal Difference Learning and TD-Gammon. Commun. *ACM* **1995**, *38*, 58–68. [CrossRef]
231. Ontanon, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; Preuss, M. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Trans. Comput. Intell. AI Games* **2013**, *5*, 293–311. [CrossRef]
232. Pascanu, R.; Li, Y.; Vinyals, O.; Heess, N.; Buesing, L.; Racanière, S.; Reichert, D.; Weber, T.; Wierstra, D.; Battaglia, P. Learning model-based planning from scratch. *arXiv* **2017**, arXiv:1707.06170.
233. Nguyen, T.H.D.; Hsu, D.; Lee, W.S.; Leong, T.Y.; Kaelbling, L.P.; Lozano-Perez, T.; Grant, A.H. CAPIR: Collaborative action planning with intention recognition. In Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 10–14 October 2011; pp. 61–66.
234. Künsch, H.R. Particle filters. *Bernoulli* **2013**, *19*, 1391–1403. [CrossRef]
235. Gordon, N.J.; Salmond, D.J.; Smith, A.F.M. Novel approach to nonlinear and linear Bayesian state estimation. *IEEE Proc. F Radar Signal Process.* **1993**, *140*, 107–113. [CrossRef]
236. Silver, D.; Veness, J. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*; Lafferty, J.D., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2010; pp. 2164–2172.
237. Macindoe, O.; Kaelbling, L.P.; Lozano-Pérez, T. POMCoP: Belief Space Planning for Sidekicks in Cooperative Games. In Proceedings of the 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 8–12 October 2012; pp. 38–43.
238. Brown, N.; Sandholm, T. Superhuman AI for multiplayer poker. *Science* **2019**, *365*, 885–890. [CrossRef]
239. Liebana, D.P.; Samothrakis, S.; Lucas, S.M.; Rohlfshagen, P. Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games. In Proceedings of the Genetic and Evolutionary Computation Conference, Amsterdam, The Netherlands, 6–10 July 2013; pp. 351–358. [CrossRef]
240. Gaina, R.D.; Lucas, S.M.; Perez-Liebana, D. Rolling Horizon Evolution Enhancements in General Video Game Playing. In Proceedings of the IEEE Conference on Computational Intelligence and Games, New York, NY, USA, 22–25 August 2017; pp. 88–95. [CrossRef]
241. Perez-Liebana, D.; Stephenson, M.; Gaina, R.D.; Renz, J.; Lucas, S.M. Introducing Real World Physics and Macro-Actions to General Video Game AI. In Proceedings of the IEEE Conference on Computational Intelligence and Games, New York, NY, USA, 22–25 August 2017; pp. 248–255. [CrossRef]
242. Justesen, N.; Mahlmann, T.; Risi, S.; Togelius, J. Playing Multi-Action Adversarial Games: Online Evolutionary Planning versus Tree Search. *IEEE Trans. Games* **2018**, *10*, 281–291. [CrossRef]

243. Justesen, N.; Risi, S. Continual Online Evolutionary Planning for In-Game Build Order Adaptation in StarCraft. In Proceedings of the Genetic and Evolutionary Computation Conference, Berlin Germany, 15–19 July 2017; pp. 187–194. [CrossRef]
244. Quadflieg, J.; Preuss, M.; Kramer, O.; Rudolph, G. Learning the Track and Planning Ahead in a Car Racing Controller. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Dublin, Ireland, 18–21 August 2010; pp. 395–402. [CrossRef]
245. Clark, C.; Fleshner, A. Fast Random Genetic Search for Large-Scale RTS Combat Scenarios. In Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Snowbird, UT, USA, 5–9 October 2017; pp. 165–171.
246. Trusty, A.; Ontañón, S.; Ram, A. Stochastic Plan Optimization in Real-Time Strategy Game. In Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 22–24 October 2008; pp. 126–131.
247. Watter, M.; Springenberg, J.T.; Boedecker, J.; Riedmiller, M. Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*; Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; pp. 2746–2754.
248. Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; Abbeel, P. Value Iteration Networks. In Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; pp. 4949–4953. [CrossRef]
249. Guez, A.; Weber, T.; Antonoglou, I.; Simonyan, K.; Vinyals, O.; Wierstra, D.; Munos, R.; Silver, D. Learning to Search with MCTSnets. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Volume 80, pp. 1817–1826.
250. Schmidhuber, J. On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models. *arXiv* **2015**, arXiv:1511.09249.
251. Botvinick, M.; Toussaint, M. Planning as inference. *Trends Cogn. Sci.* **2012**, *16*, 485–488. [CrossRef]
252. Toussaint, M.; Storkey, A. Probabilistic Inference for Solving Discrete and Continuous State Markov Decision Processes. In Proceedings of the 23rd international conference on Machine learning, Pittsburgh, PA, USA, 25–29 June 2006; pp. 945–952. [CrossRef]
253. Baier, H.; Cowling, P.I. Evolutionary MCTS for Multi-Action Adversarial Games. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Maastricht, The Netherlands, 14–17 August 2018; pp. 253–260. [CrossRef]
254. Guo, X.; Singh, S.; Lewis, R.; Lee, H. Deep Learning for Reward Design to Improve Monte Carlo Tree Search in ATARI Games. In Proceedings of the International Joint Conference on Artificial Intelligence, New York, NY, USA, 9–15 July 2016; pp. 1519–1525.
255. Perez, D.; Samothrakis, S.; Lucas, S. Knowledge-based Fast Evolutionary MCTS for General Video Game Playing. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 68–75. [CrossRef]
256. Robles, D.; Rohlfshagen, P.; Lucas, S.M. Learning Non-Random Moves for Playing Othello: Improving Monte Carlo Tree Search. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Seoul, Korea, 31 August–3 September 2011; pp. 305–312. [CrossRef]
257. Ontañón, S. Informed Monte Carlo Tree Search for Real-Time Strategy games. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Santorini, Greece, 20–23 September 2016; pp. 70–77. [CrossRef]
258. Swiechowski, M.; Tajmajer, T.; Janusz, A. Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Maastricht, The Netherlands, 14–17 August 2018; pp. 445–452. [CrossRef]
259. Alhejali, A.M.; Lucas, S.M. Using Genetic Programming to Evolve Heuristics for a Monte Carlo Tree Search Ms Pac-Man Agent. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Niagara Falls, ON, Canada, 11–13 August 2013; pp. 65–72. [CrossRef]
260. Lucas, S.M.; Samothrakis, S.; Pérez, D. Fast Evolutionary Adaptation for Monte Carlo Tree Search. In *Applications of Evolutionary Computation. EvoApplications 2014. Lecture Notes in Computer Science*; Esparcia-Alcázar, A.I., Mora, A.M., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8602, pp. 349–360. ISBN 9783662455227.

261. Vodopivec, T.; Šter, B. Enhancing Upper Confidence Bounds for Trees with Temporal Difference Values. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 278–285. [CrossRef]
262. Gelly, S.; Silver, D. Combining Online and Offline Knowledge in UCT. In Proceedings of the 24th international conference on Machine learning, Corvalis, OR, USA, 20–24 June 2007; pp. 273–280. [CrossRef]
263. Sutton, R.S. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *ACM SIGART Bull.* **1991**, *2*, 160–163. [CrossRef]
264. Sutton, R.S. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In Proceedings of the 7th International Conference on Machine Learning, Austin, TX, USA, 21–23 June 1990; pp. 216–224. [CrossRef]
265. Moore, A.W.; Atkeson, C.G. Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time. *Mach. Learn* **1993**, *13*, 103–130. [CrossRef]
266. Zhao, G.; Tatsumi, S.; Sun, R. RTP-Q: A Reinforcement Learning System with Time Constraints Exploration Planning for Accelerating the Learning Rate. *IEEE Trans. Fundam. Electron. Commun. Comput. Sci.* **1999**, *E82–A*, 2266–2273.
267. Silver, D. Reinforcement Learning and Simulation-Based Search in Computer Go. Ph.D. Thesis, University of Alberta, Edmonton, AB, Cannada, 2009.
268. Silver, D.; van Hasselt, H.; Hessel, M.; Schaul, T.; Guez, A.; Harley, T.; Dulac-Arnold, G.; Reichert, D.; Rabinowitz, N.; Barreto, A.; et al. The Predictron: End-To-End Learning and Planning. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 3191–3199.
269. Silver, D.; Sutton, R.S.; Müller, M. Temporal-difference search in computer Go. *Mach. Learn* **2012**, *87*, 183–219. [CrossRef]
270. Vodopivec, T.; Samothrakis, S.; Ŝter, B. On Monte Carlo Tree Search and Reinforcement Learning. *J. Artif. Intell. Res.* **2017**, *60*, 881–936. [CrossRef]
271. Sutton, R.S.; Precup, D.; Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* **1999**, *112*, 181–211. [CrossRef]
272. Bacon, P.-L.; Harb, J.; Precup, D. The Option-Critic Architecture. In Proceedings of the 31st AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 1726–1734.
273. Murdock, W.W.; Goel, A.K. Meta-Case-Based Reasoning: Using Functional Models to Adapt Case-Based Agents. In *Case-Based Reasoning Research and Development. ICCBR 2001. Lecture Notes in Computer Science*; Aha, D.W., Watson, I., Eds.; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2080, pp. 407–421, ISBN 978-3-540-44593-7.
274. Ulam, P.; Goel, A.; Jones, J. Reflection in Action: Model-Based Self-Adaptation in Game Playing Agents. In Proceedings of the AAAI Workshop. Technical Report WS-04-04, San Jose, CA, USA, 25–26 July 2014; pp. 86–90.
275. Arrabales, R.; Ledezma, A.; Sanchis, A. Towards Conscious-like Behavior in Computer Game Characters. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Milano, Italy, 7–10 September 2009; pp. 217–224. [CrossRef]
276. Arrabales, R.; Ledezma, A.; Sanchis, A. CERA-CRANIUM: A Test Bed for Machine Consciousness Research. In Proceedings of the International Workshop on Machine Consciousness, Hong Kong, China, 11–14 June 2009.
277. Arrabales, R.; Ledezma, A.; Sanchis, A. Establishing a Roadmap and Metrics for Conscious Machines Development. In Proceedings of the 8th IEEE International Conference on Cognitive Informatics, Hong Kong, China, 15–17 June 2009; pp. 94–101. [CrossRef]
278. Maass, W. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Netw.* **1997**, *10*, 1659–1671. [CrossRef]
279. Wang, G.; Zeng, Y.; Xu, B. A Spiking Neural Network Based Autonomous Reinforcement Learning Model and Its Application in Decision Making. In *Advances in Brain Inspired Cognitive Systems. BICS 2016. Lecture Notes in Computer Science*; Liu, C., Hussain, A., Luo, B., Tan, K., Zeng, Y., Zhang, Z., Eds.; Springer: Cham, Switzerland, 2016; Volume 10023, pp. 125–137. ISBN 978-3-319-49685-6.
280. Friedrich, J.; Lengyel, M. Goal-Directed Decision Making with Spiking Neurons. *J. Neurosci.* **2016**, *36*, 1529–1546. [CrossRef] [PubMed]

281. Funge, J.; Tu, X.; Terzopoulos, D. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA, 8–13 August 1999; pp. 29–38. [CrossRef]
282. Tadepalli, P. Lazy Explanation-Based Learning: A Solution to the Intractable Theory Problem. In Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1, Detroit, MI, USA, 20–25 August 1989; pp. 694–700.
283. Thompson, T.; Levine, J. Realtime Execution of Automated Plans using Evolutionary Robotics. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Milano, Italy, 7–10 September 2009; pp. 333–340. [CrossRef]
284. Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; Blythe, J. Integrating Planning and Learning: The PRODIGY Architecture. *J. Exp. Theor. Artif. Intell.* **1995**, *7*, 81–120. [CrossRef]
285. Jiménez, S.; Fernández, F.; Borrajo, D. The PELA Architecture: Integrating Planning and Learning to Improve Execution. In Proceedings of the 23rd national conference on Artificial intelligence, Chicago, IL, USA, 13–17 July 2008; pp. 1294–1299.
286. Beal, D.F.; Smith, M.C. Temporal difference learning applied to game playing and the results of application to shogi. *Theor. Comput. Sci.* **2001**, *252*, 105–119. [CrossRef]
287. 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* **2017**, *550*, 354–359. [CrossRef]
288. Anthony, T.; Tian, Z.; Barber, D. Thinking Fast and Slow with Deep Learning and Tree Search. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*; Guyon, I., Luxburg, U., von Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; pp. 5360–5370.
289. Perez-Liebana, D.; Lucas, S.M.; Gaina, R.D.; Togelius, J.; Khalifa, A.; Liu, J. *General Video Game Artificial Intelligence*; Morgan & Claypool: San Rafael, CA, USA, 2019; p. 191.
290. Bravi, I.; Khalifa, A.; Holmgård, C.; Togelius, J. Evolving Game-Specific UCB Alternatives for General Video Game Playing. In *Applications of Evolutionary Computation. EvoApplications 2017. Lecture Notes in Computer Science*; Squillero, G., Sim, K., Eds.; Springer: Cham, Switzerland, 2017; pp. 393–406. [CrossRef]
291. Ilhan, E.; Etaner-Uyar, A.S. Monte Carlo Tree Search with Temporal-Difference Learning for General Video Game Playing. In Proceedings of the IEEE Conference on Computational Intelligence and Games, New York, NY, USA, 22–25 August 2017; pp. 317–324. [CrossRef]
292. Van Seijen, H.; Mahmood, A.R.; Pilarski, P.M.; Machado, M.C.; Sutton, R.S. True Online Temporal-Difference Learning. *J. Mach. Learn. Res.* **2016**, *17*, 1–40.
293. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]
294. Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *arXiv* **2019**, arXiv:1911.08265.
295. Keehl, O.; Smith, A.M. Monster Carlo 2: Integrating Learning and Tree Search for Machine Playtesting. In Proceedings of the IEEE Conference on Games, London, UK, 20–23 August 2019; pp. 959–966. [CrossRef]
296. Keehl, O.; Smith, A.M. Monster Carlo: An MCTS-based Framework for Machine Playtesting Unity Games. In Proceedings of the IEEE Conference on Computatonal Intelligence and Games, Maastricht, The Netherlands, 14–17 August 2018; pp. 141–148. [CrossRef]
297. Sigurdson, D.; Bulitko, V. Deep Learning for Real-Time Heuristic Search Algorithm Selection. In Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Snowbird, UT, USA, 5–9 October 2017; pp. 108–114.
298. Quitério, J.; Prada, R.; Melo, F.S. A Reinforcement Learning Approach for the Circle Agent of Geometry Friends. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Tainan, Taiwan, 31 August–2 September 2015; pp. 423–430. [CrossRef]
299. Chen, Y.; Li, Y. Macro and Micro Reinforcement Learning for Playing Nine-ball Pool. In Proceedings of the IEEE Conference on Games, London, UK, 20–23 August 2019; pp. 258–261. [CrossRef]

300. Tastan, B.; Chang, Y.; Sukthankar, G. Learning to Intercept Opponents in First Person Shooter Games. In Proceedings of the IEEE Conference on Computational Intelligence and Games, Granada, Spain, 11–14 September 2012; pp. 100–107. [CrossRef]
301. Barriga, N.A.; Stanescu, M.; Buro, M. Combining Strategic Learning and Tactical Search in Real-Time Strategy Games. In Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Snowbird, UT, USA, 5–9 October 2017; pp. 9–15.
302. Archibald, C.; Altman, A.; Shoham, Y. Analysis of a Winning Computational Billiards Player. In Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, CA, USA, 11–17 July 2009; pp. 1377–1382.
303. Hefny, A.S.; Hatem, A.A.; Shalaby, M.M.; Atiya, A.F. Cerberus: Applying Supervised and Reinforcement Learning Techniques to Capture the Flag Games. In Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference, Palo Alto, CA, USA, 22–24 October 2008; pp. 179–184.