algorithms

Review

Techniques and Paradigms in Modern Game AI Systems

Yunlong Lu and Wenxin Li *

School of Computer Science, Peking University, Beijing 100871, China; luyunlong@pku.edu.cn
* Correspondence: lwx@pku.edu.cn; Tel.: +86-10-62753425

Abstract: Games have long been benchmarks and test-beds for AI algorithms. With the development of AI techniques and the boost of computational power, modern game AI systems have achieved superhuman performance in many games played by humans. These games have various features and present different challenges to AI research, so the algorithms used in each of these AI systems vary. This survey aims to give a systematic review of the techniques and paradigms used in modern game AI systems. By decomposing each of the recent milestones into basic components and comparing them based on the features of games, we summarize the common paradigms to build game AI systems and their scope and limitations. We claim that deep reinforcement learning is the most general methodology to become a mainstream method for games with higher complexity. We hope this survey can both provide a review of game AI algorithms and bring inspiration to the game AI community for future directions.

Keywords: game AI; supervised learning; reinforcement learning; multi-agent learning

1. Introduction

The ultimate goal of artificial intelligence is to reach human-level on a wide range of tasks. Turing test [1] proposed by Alan Turing in 1950 is the earliest criterion to test a machine's ability to exhibit intelligent behavior as humans, which is controversial because a specially designed AI can meet such standards by targeted imitation or deception instead of real intelligence. However, games are born to test and challenge human intelligence, which can be excellent benchmarks for the cognitive and decision-making abilities of AI systems. The diversity of games has provided a rich context for gradual skill progression in the development of artificial intelligence. AI systems beating professional humans in games with increasing complexity have always been considered milestones of AI technologies.

With the boost of computational power and the application of new algorithms, AI systems have made great strides to play games that were once considered exclusively mastered by humans because of their complexity. Since AlphaGo beat professional human players in Go [2], many milestones have been reported in various games, from board games like Go [2–4] and card games like Texas Hold'em [5–7] to video games like StarCraft [8], Dota 2 [9] and HoK [10]. These AI systems achieve superhuman performance in games with increasing complexity while using a wide range of AI techniques. It seems that different approaches are chosen based on the characteristics of the games, but it is hard to find the pattern of algorithm selection when the algorithms vary in each system.

In this survey, we provide a systematic review of the AI techniques typically used to create game-playing agents and summarize how they are used in recent AI milestones. We show that a typical AI system can be broken into basic components related to specific features or challenges of the games they tackle. By analyzing the choices of the algorithms and the characteristics of the games, we extract three kinds of paradigms to build AI systems for different types of games. The application scope and limitations of each paradigm are further discussed, as an indication of the general method applicable to games of higher complexity, such as real-world sports games.

check for updates

Citation: Lu, Y.; Li, W. Techniques and Paradigms in Modern Game AI Systems. *Algorithms* **2022**, *15*, 282. https://doi.org/10.3390/a15080282

Academic Editor: Akemi Galvez Tomida

Received: 13 July 2022
Accepted: 9 August 2022
Published: 12 August 2022

Some related works also investigate recent milestones of game AI. Risi et al. [11] only gives a general introduction of the categories of games and AI technologies and focuses on the application of these techniques to areas outside of game playing, like content generation and player modeling. Perhaps Yin et al. [12] is the most similar work to us, which also makes the comparison between game AI systems and discusses general paradigms. However, that survey mainly focuses on the different training frameworks in various types of games. It does not summarize the basic algorithms used in these frameworks nor compare them concerning game features. Instead, this survey analyzes the basic algorithmic components used in these milestones and summarizes both the techniques and paradigms based on the features of games.

The rest of this survey is organized as follows. Section 2 discusses the background of AI applications in creating game-playing agents. A brief history of game AI milestones is listed, as well as some typical features of games and the modeling of games in the context of AI algorithms. In Section 3, many game AI algorithms are presented as the basic components of game AI systems, which are categorized based on their mechanics and the game features they tackle. Section 4 summarizes the implementation of recent state-of-the-art game AI systems. Section 5 makes a comprehensive comparison of these milestones by decomposing each of them into components and relating the choices to the characteristics of the games. We further extract the common paradigms of these milestones and discuss the general method of game AI as future trends.

2. Background

Recent years have witnessed remarkable progress in artificial intelligence, with games often serving as benchmarks. While board games have been the focus of AI research since the beginning of this field, the advance in algorithms has drawn attention to increasingly complex games in the last decade, such as card games and video games. These games have features that challenge game AI research, spawning many new algorithms in the last decade. In this section, we first discuss the breakthroughs in the history of AI game playing to show how games are used as AI benchmarks. Then we summarize the key features of the games solved in recent years. Finally, we introduce the modeling of games in the context of AI algorithms as a basis for the AI techniques discussed in the next section.

2.1. Game AI Benchmarks

Since the earliest computer, ENIAC, was invented in 1945, game playing has been an important area in artificial intelligence. In 1951, Christopher Strachey wrote a checkers program, and Dietrich Prinz wrote one for chess [13], which were the earliest AI game-playing programs. Most early research on game AI was focused on classic board games like checkers and chess because they have elementary and highly constrained rules yet great complexity that have challenged humans for hundreds or even thousands of years. AI systems beating professional humans in these games have always been considered as milestones and breakthroughs in AI technologies.

The first of these milestones was TD-Gammon developed by Gerald Tesauro in 1992 [14], a backgammon program to beat professional humans. In 1994, the Chinook Checkers program beat the World Checkers Champion Marion Tinsley [15]. Perhaps the most well-known milestone was IBM's Deep Blue, a Chess program that won against reigning grandmaster Kasparov in 1997 in a very famous and publicized event [16]. The latest milestone in board games was in the game of Go. In 2016 Google Deepmind's AlphaGo program beat Lee Sedol in a five-game match [2], and in 2017 a newer version of AlphaGo won against world champion Ke Jie in a three-game competition [3]. While it is possible to construct more complex board games than Go, no such games are popular for humans.

However, classic board games are relatively easier in game AI due to their discrete turn-based mechanics, highly formalized state representation, and fully-visible information to all players. Researchers have turned to more challenging games like card games and video games in the last decade. These games are much more difficult to solve due to the large state and action space, long time horizon, information asymmetry, and possible cooperation between players. Thanks to the rapid development of computational power and AI techniques, several milestones have been achieved.

Card games often involve randomness in dealing cards and asymmetries of information between different players. In 2015, Bowling and his team solved Heads-Up Limit Hold'em, achieving an approximate optimal solution of the game [17]. Their team further built DeepStack in 2017, a program that beat professional players in Heads-Up No-Limit Hold'em [5]. In 2018 Libratus by Brown's team also won professional players with different techniques from DeepStack [6]. Their team later built Pluribus in 2019, achieving superhuman performance in six-player No-Limit Hold'em [7]. In 2019 Suphx, made by MSRA, beat most of the top human players in Riichi Mahjong [18], and in 2022 JueJong, built by Tencent AI Lab, beat a human champion in two-player Mahjong [19]. There are also efforts to solve Doudizhu, a popular card game in China, including DouZero [20] and PerfectDou [21], but no superhuman performance has yet been reported.

Video games are real-time frame-based games where players receive raw pixel-level input and take actions simultaneously. In 2014 Google DeepMind proposed DQN to play the classic Atari 2600 video games and achieved superhuman performance in some of them [22]. Multi-player Online Battle Arena (MOBA) games are complex video games that involve both cooperation and competition between players. In 2019 AlphaStar, proposed by Google DeepMind, beat professional players in StarCraft, which is the first successful AI system to achieve superhuman performance in MOBA games [8]. The same year, OpenAI built OpenAI Five to play Dota 2 and beat OG, the world champion team. It later defeated 99.4% of human players in a public online arena [9]. In 2020 Tencent AI Lab built JueWu to play Honour of Kings and won 40 of 42 matches against professional teams. It also achieves a 97.7% win rate in large-scale public tests against top players [10].

An important reason why games are excellent benchmarks for AI is that games are created to test and challenge human intelligence. Games with high quality usually act as teachers and can exercise many of our cognitive and decision-making abilities. Just as children learn about the world by playing with games and toys during their first years of life, games provide test-beds for gradual skill progression to test AI algorithms with different capabilities. Unlike narrow benchmarks in other fields of AI, the diversity of games can provide a rich context for AI algorithms. Board games, with their formalized state representation and perfect information, only require searching and planning from the current game state. Card games, with their non-deterministic transition and imperfect information, reveal more sophisticated strategies, such as bluffing and deception, skills that are normally reserved for humans. Video games, with their high-dimensional raw state and long time horizon, require feature extraction, memorization, long-term planning, and possible multi-agent cooperation. These characteristics make games strong benchmarks for the development of AI technology.

2.2. Game Features

Since AlphaGo achieved superhuman performance in the game of Go, in recent years, many popular games played by humans that were once considered impossible for AI to conquer have been solved. These games are difficult to solve because some features of them bring diverse challenges and difficulties for AI research. Here we focus on games involved in the milestones discussed in Section 4. Table 1 lists several key features of these games, which play an important role in the selection of techniques used to tackle them.

2.2.1. Real Time

Board games and card games are turn-based games, where players take turns to take action and receive new observations. These games do not have a long time horizon, and a typical episode lasts tens or hundreds of turns. Real-time planning algorithms like tree search is often used in turn-based games because the transition model of the environment is known, and seconds or even minutes of thinking time are permitted. Video games are real-time or frame-based games, where the observation is presented frame by frame at a fixed frequency, and the players can take actions at any frame. Such games have a long time horizon, and an episode can last for thousands of frames. Typical AI systems use direct network inference without planning because the environment model is unknown, and a fast response of actions is required.

Table 1. Features of the games tackled in recent milestones.

Game Types	Name	Players	Real-Time	Imperfect Information	Stochasticity	Cooperation	Heterogeneous
Board games	Go	2	✗	✗	✗	✗	✗
Card games	HUNL	2 or 6	✗	✓	✓	✗	✗
	Riichi Mahjong	4	✗	✓	✓	✗	✗
	1-on-1 Mahjong	2	✗	✓	✓	✗	✗
	Doudizhu	3	✗	✓	✗	✓	✗
Video games	Starcraft	2	✓	✓	✓	✗	✓
	Dota 2	10	✓	✓	✓	✓	✓
	Honour of Kings	10	✓	✓	✓	✓	✓

2.2.2. Imperfect Information

Classic board games are games with perfect information, where each player knows all the information about the current game state. According to Zermelo's theory [23], there exists an optimal solution in deterministic perfect-information games, so the purpose of strong AI is to find or approximate that optimal solution. However, most games are of imperfect information, where each player has hidden information that other players cannot observe. In such games, each player may reason about others' private information according to their past actions before making their own decision, which in turn affects others' belief in his private information. Such recursive reasoning brings uncertainty and complexity to the evaluation of strategies. Instead of an optimal solution, algorithms seek to find some equilibrium, such as Nash equilibrium, in imperfect-information games. Nash equilibrium [24] is a solution concept where each player cannot get a higher payoff if he changes his policy only. It also minimizes the exploitability of each player, which is defined as the scoring points one will lose when faced with the worst opponents.

2.2.3. Stochasticity

In games like Go and Doudizhu, the transition of the environment is deterministic, which means that the same initial state and action sequence will lead to the same episode. In contrast, in most games, random events like dice rolling or card dealing bring stochasticity. It is worth noting that stochasticity does not always lead to imperfect information and vice versa. For example, in Texas Hold'em, the card faced down and dealt to each player introduces hidden information, while the dealing of public cards only introduces stochasticity because all players can observe that. Stochasticity in the transition model brings extra complexity to real-time planning algorithms to explore the branches of chance nodes and higher variance for learning algorithms to converge because the same action sequence may have very different payoff values.

2.2.4. Cooperation

Most board games and card games are zero-sum games that are purely competitive. The player who seeks to maximize his payoff also reduces other players' payoff. An exception is Doudizhu, where two peasants cooperate against one landlord and receive the same payoff. MOBA games are usually team games with cooperation, where the competition happens between two teams of players. Starcraft only involves two players who build facilities and direct an army to fight, which is not a cooperative game. Cooperation brings greater challenges to AI algorithms because mechanisms of communication and credit assignment have to be designed to motivate agents for the same target, and some agents have to sacrifice their own interests for team benefits.

2.2.5. Heterogeneous

In board games and card games, different players are homogeneous agents that share the same state space and action space. Although the strategies may differ for players in different positions, they share the same understanding of the game rules. However, many MOBA games are designed to have heterogeneous agents that have quite different strategies and action space. For example, in Starcraft, each player chooses one of three races with different mechanics. In Dota 2 and Honour of Kings, each player selects a hero from a hero pool with unique skills and game-playing strategies. The setting of heterogeneous agents introduces great complexity for AI algorithms as they need to learn different policies under each race or hero combination.

2.3. Game AI Modeling

As an important field of artificial intelligence, game playing is quite different from other fields like computer vision (CV) and natural language processing (NLP) because it can offer the richest form of human-computer interaction. CV and NLP are usually considered as cognitive intelligence, where computers try to extract useful information from images, texts, and videos to understand and interpret them. However, in the context of game playing, computer agents need to constantly interact with the environment and achieve specific goals through the actions they choose. In this way, game AI belongs to the category of decision intelligence, and this process of interaction between agents and environments is the core of game AI modeling.

In general, when modeling a game, the interaction between agents and the environment is divided into discrete steps. In each step, the agents to act first receive observations from the environment, and each chooses an action. Under these actions, the environment will transit to a new state and present new observations for these agents in the next step. For each agent, the intelligence lies in its policy or strategy, which is a mapping from historical observation sequences to actions to decide what to do at each time step. Such a policy model is the ultimate learning target of all AI systems to play specific games, though many algorithms also learn a value model to evaluate the expected payoff of game states and choose actions to maximize the value.

In the context of reinforcement learning, single-agent games are typically modeled as Markov Decision Processes (MDPs) [25]. In each time step, the environment is in some state, and the agent must choose an action available under the state. The environment responds by moving into a new state and giving the agent a corresponding reward. The transition probability and reward satisfy the Markov property that they are only related to the current state and action. For environments that do not follow the Markov property, the state can be defined as the historical observation sequence which satisfies the property. In games like multi-armed bandits, each game only lasts for one step, and the target is to maximize the expected payoff of each game. For games with a longer time horizon or infinite steps, the agent's target is to maximize its cumulative reward in the future. A decaying factor γ is defined to balance the trade-off between the reward in the next step and future steps.

When it comes to multi-agent settings, things become more complex because different agents can have separate observation spaces, causing information asymmetry. These games are usually modeled as Partially Observable MDP (POMDP), where the observations are distinguished from game states. In addition to the MDP defined on hidden states, there is a mapping from these states to observations for each agent, which describes the emitting probability of observations under each hidden state. Since agents have to infer the current state from historical observation sequences, the Markov property is not satisfied on observations, and the strategy has to be defined on the whole sequence of historical observations and actions.

Game theory pays more attention to the interaction between multiple agents rather than the transition dynamics of the environment, so it models games from a different perspective [26]. The extensive form is a straightforward way to model games in the context of game theory. It models the game as a tree, where each node is a game state and a decision point of a player. Edges from each node represent the available actions of the player to act, and the successor node is the next game state after the action is performed. Each agent's payoff is marked on leaf nodes, representing the end of the game. Information sets are defined as the set of states that an agent cannot distinguish because of hidden information, and policies are defined as mappings from information sets to actions. Another way of formally describing games is normal-form representation, which uses a multi-dimensional matrix to list the policy space of each agent and the payoff under each strategy profile, i.e., the combination of strategies of each agent. This form of representation models a situation where players select complete action plans simultaneously, but suffers from an exponential size of strategy space in games with long time horizons.

Reinforcement learning and the game theory model games as a process of interaction from different perspectives. Reinforcement learning models games as a control problem and seeks to maximize the payoff of individual agents in the interaction with an environment, which is initially designed for single-agent games. However, in multi-agent settings, the strategies of different agents can affect each other, resulting in unstable dynamics of the environment from the perspective of individual agents. Game theory puts more emphasis on the interaction between agents. By directly modeling the game as the payoffs under different action sequences or strategy profiles, such representations better capture the interactive nature of multi-agent games and help analyze different solution concepts instead of a static optimal solution.

3. Game AI Techniques

Many AI techniques have been proposed to build game-playing agents for different games. Figure 1 shows a timeline of different types of techniques and the type of games they tackle. In terms of algorithms, evolutionary methods are one of the earliest ways to create game-playing agents by randomized search and selection in the parameter space. Reinforcement learning models games as control problems and seeks to maximize cumulative rewards of individual agents in the interaction with the environment. Given the transition model of the environment, real-time planning algorithms expand a search tree and calculate the best action when each state is encountered during the gameplay, as an enhancement to the original policy and value model. Regret-based methods deal with multi-agent problems and approximate Nash equilibrium by playing games repeatedly and minimizing the cumulative regrets of each player. Self-play methods are game-theoretic methods to calculate Nash equilibrium in multi-agent games, which in practice can extend RL algorithms to multi-agent settings. Besides, there are also RL algorithms specially designed for multi-agent settings by decomposing value functions [27,28] or following a centralized-training-decentralized-execution (CTDE) framework [29,30].

From the timeline, we can see that in different times of game AI history, the increasing complexity of games has spawned new types of algorithms designed for specific features of games. Earlier methods are focused on single-agent problems and try to find an optimal solution for them. As randomized global optimization algorithms, evolutionary methods can solve small-scale problems but suffer from low performance and inefficiency when the policy space is too large. Reinforcement learning initially deals with problems whose transition models are known, using model-based methods like value iteration and policy iteration. Modern RL algorithms are mostly model-free methods that do not require explicit transition dynamics and directly learn policy or value models through interactions with environments. Traditional planning methods like Minimax and MCTS are designed for games with perfect information, while in the last decade, researchers have turned to more complex games such as card games and video games, which are all games with imperfect information. Real-time planning has been applied to these games using variants of MCTS or new algorithms like continual re-solving. There are also methods specially designed for multi-agent games with imperfect information, like CFR algorithms, self-play schemes, and multi-agent RL algorithms.

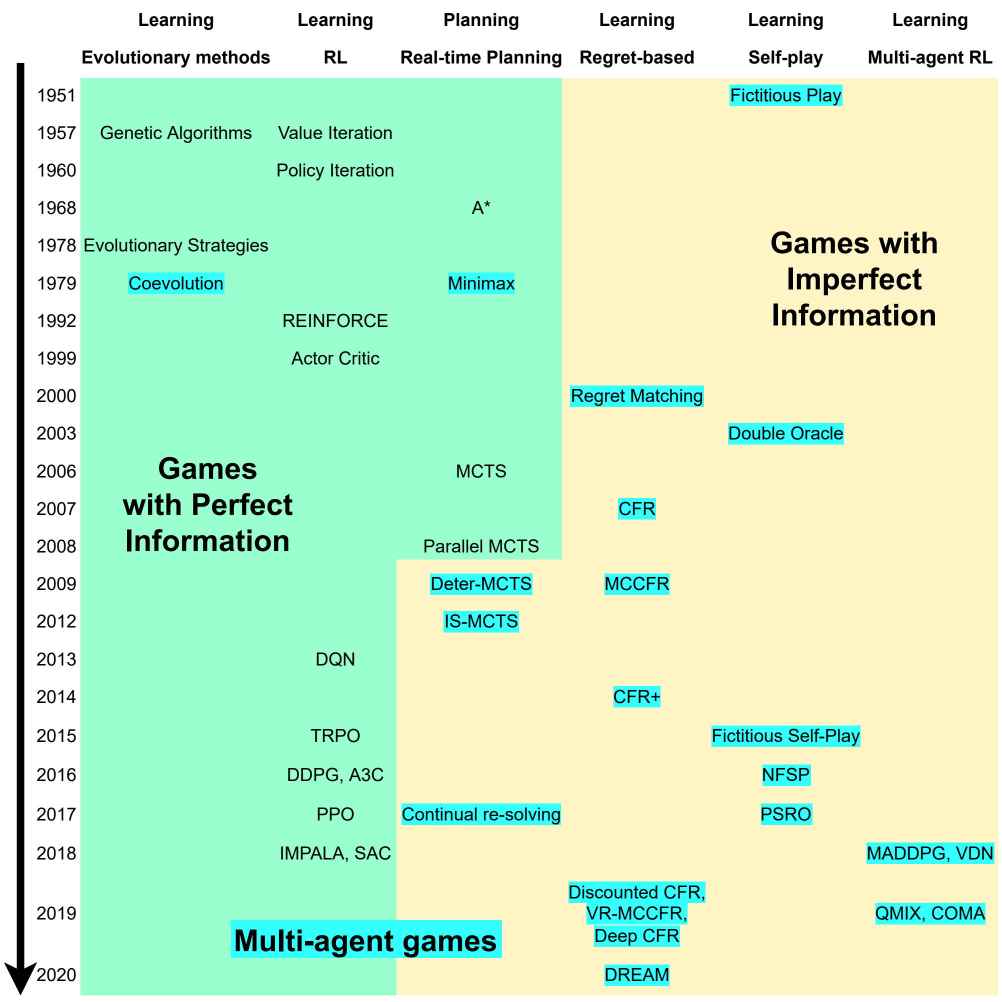

Figure 1. Timeline of different AI techniques to build game-playing agents. Colors indicate the type of games each algorithm tackles, green for games with perfect information, yellow for games with imperfect information, and blue for multi-agent games.

Meanwhile, as the games become more complex, modern game AI systems are no longer application of a single algorithm but a combination of multiple techniques. A typical AI to play a specific game usually involves some 'prior' knowledge, either explicitly incorporated by human experts or learned through pre-training phases. Such knowledge is then used in the game-play, combined with real-time planning and reasoning. This section further categorizes various AI techniques from this perspective. By discussing real-time planning and learning algorithms separately, it would be easier to break modern game AI systems into basic components and compare them thoroughly in the following sections.

3.1. Real-Time Planning

In most cases, the state space of a game is so large that it is difficult to have an optimal strategy for every possible state before the game starts. If the state transition model of the game is known, planning can be done when each specific game state is encountered during the gameplay, as a computation whose output is the selection of a single action under the current state. Such methods are called real-time planning, or decision-time planning, and are most useful in applications where fast responses are not required. For example, in a chess-playing program, seconds or even minutes of thinking time is permitted and can be used for planning, while in a real-time strategy game, low latency for action selection is the priority, and planning is usually not used.

When an evaluation function of game states is available, the simplest form of planning is to choose an action by comparing the values of model-predicted next states for each action. Such planning can look much deeper than one step ahead and expand a search tree to evaluate many future states under different sequences of actions, leading to a more far-sighted agent. These methods are generally called heuristic search. One of the most famous algorithms is A* [31], which uses an evaluation function to guide the selection of unexplored nodes. In general, searching deeper usually yields better policies when the evaluation function is not perfect because it eliminates some errors of value estimation by looking ahead for some steps. However, searching deeper costs more computation, and the search depth is usually fixed or decided by iterative deepening to limit the time cost in practice.

Minimax [32] is a classical real-time planning algorithm in two-player competitive settings and is widely used in board games. The basic assumption is that each player wants to maximize his own values. When expanding the search tree, nodes are divided into max-nodes and min-nodes based on which player is to act. Once the depth limit is reached, an evaluation function on game states is utilized to estimate the value of the first player. Alpha-beta pruning can be further applied in that the value of some subtrees makes no difference to the value of their parent nodes under the min-max mechanism. The Minimax algorithm can be generalized to multiplayer settings, where each node maximizes the value of the player to act, and the evaluation function at leaf nodes returns the values of all players.

Monte-Carlo tree search is another real-time planning algorithm that achieves massive success in board games. It is mainly responsible for the progress achieved in computer Go from a weak amateur level in 2005 to a grandmaster level in 2015 [25]. At each state encountered, the algorithm simulates many trajectories starting from the current state and running to a terminal state. Each simulation first selects a path in the search tree using a tree policy, expands a leaf node, and plays to the end of the episode with a rollout policy. The final score is used to update the state-action value along the path. The main idea is to extend the initial portions of trajectories and focus on those that have received higher evaluations from earlier simulations. Typically, the rollout policy is chosen to be simple to reduce the time cost of each simulation. The tree policy needs to balance exploration and exploitation so that the search is mainly focused on promising trajectories while not missing out on potentially better moves in the unexplored parts. In 2006 Kocsis proposed UCT [33], which applies upper confidence bounds in MCTS to achieve maximum expected payoffs.

There are many variants of MCTS to improve its efficiency and performance. P-UCT [34] combines prior knowledge into MCTS by using default rollout policy and incorporating evaluation function into tree policy. Rapid action value estimation (RAVE) [35] assumes that the value of an action is similar under different states and estimates the value of actions based on all simulations starting from the root state. It allows value estimation to generalize across subtrees to achieve higher efficiency. Parallel MCTS [36] proposes three ways to execute MCTS simulations simultaneously by using the different granularity of parallelization: leaf parallelization runs multiple simulations from the same leaf, root parallelization uses multiple search trees and tree parallelization runs simulations from root node in the same tree. Experiments show that root parallelization achieves not only higher efficiency because the trees are not shared across processes, but also better performance, possibly because using multiple trees can get rid of local minima.

MCTS can also be applied to imperfect-information games. One popular approach is determinization, which has achieved success in games like Bridge [37] and Klondike Solitaire [38]. The main idea is to simply sample states from the current information set and fix the outcomes of future chance events, making instances of deterministic games equivalent to the original one. However, this approach searches a tree of game states that does not truly reveal the imperfect nature of the game and suffers from two problems of strategy fusion and non-locality [39]. Another approach is information set MCTS (IS-MCTS) [40], which searches trees of information sets rather than trees of game states, and can analyze the game structure more directly, reserving the variety of hidden information and stochasticity. Experiments show that IS-MCTS achieves better results than determinization MCTS on imperfect-information games such as Doudizhu [41].

Another real-time planning algorithm for imperfect-information games is called continual re-solving, which has achieved huge success in the game of no-limit Texas Hold'em [5]. It is based on CFR-D, which provides a theoretically sound framework for the decomposition and nested safe subgame solving of imperfect-information games [42]. Traditionally, imperfect-information games have been considered as a generally indivisible whole and cannot be decomposed into subgames to solve, because simply solving a subgame in such games usually produces a subgame strategy that is not part of the equilibrium solution of the whole game. CFR-D uses a player's own range and opponent counterfactual values as constraints of subgame solving to ensure the re-solved subgame strategy is not worse than the previous strategy of the whole game. Here a player's range means the opponent's belief of his private cards, and the counterfactual value is the expected payoff of the current game state, assuming the current payer tries his best to reach this state. Continual re-solving further develops this idea and combines depth-limit search trees with an evaluation function that takes players' ranges as input and produces counterfactual values as outputs. For example, Deepstack trains deep counterfactual value networks in advance and uses continual re-solving as real-time planning algorithms, achieving superhuman performance in no-limit Texas Hold'em.

3.2. Learning

Just in the same way that people do not play a new game well when they are just starting, and it takes time to get familiar with the game, most game AI systems also have a training phase to learn some prior knowledge of the game. Such prior knowledge can be stored in models, such as policy models or value models, and is used in the inference phase combined with real-time planning algorithms during actual gameplay. In most cases, this learning phase is the most fundamental part of building a game AI system and is the focus of game AI research. This section discusses various algorithms as building blocks used in modern game AI systems.

3.2.1. Evolutionary Methods

Evolutionary methods [43] are randomized global optimization algorithms inspired by the natural selection process. The basic idea is to create a population of individuals where the fitter ones have a higher probability of reproducing and inheriting part of their structures. Variation operators, including recombination and mutation, are applied to create necessary diversity within the population, and the selective pressure can increase the mean fitness of the whole population over time. This process can be viewed as if evolution is optimizing the fitness function by approaching optimal values closer, making it a strong optimization algorithm in problems where it is hard to find optimal solutions by human experts.

There are several components to specify to define a specific evolutionary algorithm. The first one is the representation or encoding of the solutions. Such encoding can simplify the odd solution space in the problem context into the space of genes so that the variation operators can be mathematically defined. Since one does not know in advance what the optimal solution looks like, it is usually desirable that all possible feasible solutions can be represented under the encoding. Another important component is the fitness function which the population should adapt to improve. In the context of game AI, the solution space usually refers to policy space, and the fitness function is often chosen as the performance of a policy in the game, like the score or payoff it can receive. The implementation of variation operators, parent selection schemes, and survival selection schemes also varies in different evolutionary algorithms.

The most well-known evolutionary algorithm variants are genetic algorithm (GA) and evolutionary strategies (ES) [44]. GA encodes solutions as binary strings and defines variation operators as one-bit flip and crossover of binary strings. The probability of parent selection is proportional to the fitness function, and the spawned children replace all parents immediately. ES encodes solutions as vectors of floating-point values and defines variation operators as Gaussian perturbation and interpolation between vectors. The parents are uniformly selected, and the individuals with the highest fitness function in both parents and children survive.

When applied to multi-agent games, coevolution [45] is a popular evolutionary approach that simulates the interactive nature of the multi-agent settings. The core idea is to define the fitness function as the relative scores received by agents fighting against each other instead of their absolute performance when interacting with the environment. In practice, the algorithm can either use a single population and have individuals challenge each other to evaluate their fitness function or create multiple populations and exploit an arms race, having individuals from different populations battle. It is assumed that such competitive coevolution can improve the fitness of species in a highly interactive environment, just as coevolution does in the natural world. Algorithms involving coevolution have achieved success in many games, including Tic-Tac-Toe [46], Pursuit and Evasion [47], Predator and Prey [48], real-time strategy games like Capture The Flag [49] and Planet Wars [50], and a collectible card game called Hearthstone [51].

3.2.2. Supervised Learning

Supervised learning is a data-driven method to approximate the underlying function between data and their attributes. In the context of game AI, the data usually refers to the game states or observations, and the task is to learn a policy model or value model that predicts the action to choose or the estimated value under the current state. Such algorithms require lots of labeled data in the form of state-action or state-value pairs, usually collected from human data of gameplay or data generated by other game-playing algorithms. Once a policy or value model is trained, it can be used as prior knowledge in the inference stage, combined with some real-time planning algorithms.

In general, supervised learning trains an approximating function by modifying the parameters in the function model. There are many ways to represent such functions, like support vector machines, decision trees, and artificial (deep) neural networks, each with a different algorithm to modify the parameters. Here we focus on the modern approach of neural networks, though in some scenarios, a classic method like decision trees is preferable due to its interpretability. Most modern game AI systems use neural networks to represent policy or value function due to their strong expressivity [52] and adaptation of feature extraction. Variants of neural networks like convolutional neural network (CNN) and recurrent neural network (RNN) are popular due to their ability to extract spatial and temporal features respectively.

The application of supervised learning in modern game AI systems can be divided into three types according to the data source. The most common one is to use human data. Using supervised learning on human data can learn implicit human knowledge and store it in policy models or value models. However, even if the model achieves perfect accuracy on the training set, the generalization error of the model is inevitable, so the level of the model usually cannot surpass the level of the humans it imitates. Besides, models learned from human data are likely to be misled and fall into the traps of human strategy, especially in those complex games where the strategies of professional humans may be far from optimal. Such models are usually used as an initialization [2,8,18] to other learning algorithms like reinforcement learning as a warm start, which can speed up the training during the early phase.

There are two other types of supervised learning applications where the data source is not from human players. One of them is knowledge extraction. Suppose there is a game-playing algorithm that runs too slowly to be used in real-time inference but can be used to generate unlimited data offline; supervised learning can be used to train a model that extracts the implicit knowledge hidden in these data but takes much less time to inference. For example, DeepStack [5] trains three value models at different stages of the NLTH game by supervised learning on data generated by continual re-solving combined with the value model at the next game stage. These models store the knowledge of state value estimation at different game stages and can be used in real-time inference. The other one is knowledge distillation [53], where a lightweight unified model is trained to clone the behavior of another large model. Instead of using the ground-truth label as hard targets in knowledge extraction, knowledge distillation uses probability distribution as soft targets and adopts Shannon entropy as the loss function to reserve the generalization ability of the original model. For example, JueWu [10] uses supervised learning to learn a unified student model from data generated by multiple teacher models with fixed hero combinations, to extract the unified strategy with arbitrary hero combinations in the game Honor of Kings.

3.2.3. Reinforcement Learning

Reinforcement learning is the area of machine learning that studies how agents can take actions in an environment to maximize cumulative rewards. Different from supervised learning where the target is to learn a mapping of labeled data pairs, reinforcement learning deals with control problems and learns how to map situations to actions. Actions are not given as ground truth, but the learner needs to discover which actions can lead to higher future rewards by interacting with the environment and trying them out. The learning focus is usually on the balance of exploration (for potentially better actions) and exploitation (to maximize cumulative rewards). Reinforcement learning has become one of the most popular methods in the field of game AI because learning how to play a game is itself a control problem, which can be directly modeled in the setting of reinforcement learning.

Typically, reinforcement learning models the environment as a Markov Decision Process (MDP). In each time step, the environment is in some state, and the agent must choose an action available under the state. The environment responds by moving into a new state and giving the agent a corresponding reward. The transition probability and reward satisfy the Markov property that they are only related to the current state and action. When the transition model is known, generalized policy iteration uses dynamic programming to solve the optimal policy and its value function, defined as the expected cumulative rewards of states or state-action pairs. One of the most common variants is value iteration, which is based on Bellman Equation that describes the relationship between the policy and value function.

Value iteration is a model-based algorithm because it requires the complete model of MDP. However, in most cases, the environment model is unknown, and model-free algorithms are preferable, which learn from experiences by interacting with the environment. There are two kinds of model-free algorithms, value-based and policy-based. Value-based algorithms optimize the policy by approximating the values of states or state-action pairs and selecting better actions based on these values. For environments with finite state space, the value function can be represented using arrays indexed by states. These kinds of algorithms are called tabular methods [25], and there are different ways to update the value function. Monte Carlo (MC) algorithm updates the value function based on the cumulative rewards towards the end of the episode, while the temporal difference (TD) algorithm updates the value function based on the current reward and the value of the next state in a bootstrapping manner. Monte Carlo algorithm usually suffers from large variance, so algorithms using a TD target such as Q-learning are more preferred in practice. When the state space is too large to fit in memory, function approximation can be used to approximate the value function. For example, DQN [22] is the variant of Q-learning which uses deep neural networks to approximate the state-action value function and achieves success on Atari games.

Policy-based algorithms are another kind of model-free algorithms, which have become increasingly popular due to the development of deep learning. These algorithms directly learn parameterized policies based on gradients of some performance measure using the gradient descent method. One of the earliest works is REINFORCE [54], which samples full episode trajectories with Monte Carlo methods to estimate return as the loss function. However, pure policy-based algorithms suffer from high variance, and actor-critic algorithms [55] have been proposed, which use actors to learn parameterized policies and critics to learn value functions, allowing the policy updates to consider the value estimates to reduce the variance. There are many variants of actor-critic algorithms. Deep deterministic policy gradient (DDPG) [56] addresses the issue of large action space by adding sampled noise to its actor's policy, allowing more exploratory behavior. Asynchronous Advantage Actor-Critic (A3C) [57] is a distributed algorithm where multiple actors running on different threads interact with the environment simultaneously and compute gradients in a local manner. Importance Weighted Actor-Learner Architecture (IMPALA) [58] is another distributed algorithm that uses V-trace to compensate for the gradient error introduced by asynchronous execution. Trust Region Policy Optimization (TRPO) [59] and Proximal Policy Optimization (PPO) [60] are state-of-the-art policy-based algorithms, where policy changes are incorporated into the loss function by adding KL-divergence to the loss or using loss clipping to prevent abrupt changes in policies during training.

3.2.4. Multi-Agent Learning

Learning policies in multi-agent environments are very different from single-agent ones because each agent's behavior can affect other agents' observations, making the environment non-stationary from one agent's perspective. Instead of solving an optimal policy in single-agent settings, multi-agent learning aims to find some equilibrium, such as Nash equilibrium [24], where each player cannot get a higher payoff if he changes his policy. There are many algorithms specially designed for multi-agent learning, and here list some of the most important ones used in modern game AI systems.

Regret matching [61] is a simple and intuitive algorithm to solve the Nash equilibrium of normal-form games. In the algorithm, players choose their actions with probabilities proportional to measures of regret for not having chosen other actions in the past. Counterfactual regret minimization (CFR) [62] extends its application to extensive-form games, which has become a powerful tool to solve imperfect-information games. However, vanilla CFR traverses the whole game tree on each iteration and takes many iterations to converge, making it computationally expensive to solve larger games. Many variants of CFR have been proposed to improve its efficiency. CFR+ [63] and Discounted CFR [64] discount regrets from earlier iterations and reweight iterations in various ways to speed up the training. MCCFR [65] and VR-MCCFR [66] only sample a few paths when traversing the game tree, making them capable of solving games with massive game trees. Some variants utilize state abstraction [67] and function approximation [68] to reduce the time and memory cost. It was not until neural networks were used [69–72] that state abstraction and approximation could be made without human knowledge. Most of these methods with deep learning also take advantage of the ideas in previous variants, including sampling, discounting, and reweighting, resulting in better performance in complex games.

Combining single-agent RL with proper self-play techniques in competitive multi-agent environments can also approach a Nash equilibrium. The earliest algorithm is fictitious play (FP) [73] in two-player zero-sum games, where each agent calculates its best response to the opponent's average policies. Fictitious self play (FSP) [74] extends its application to extensive-form games. Neural FSP (NFSP) [75] adopts neural networks as the policy model to deal with larger games, using reinforcement learning to calculate the best response and supervised learning to learn average policies. Double oracle (DO) [76] only considers a small subset of the policy space, where each agent iteratively calculates the Nash equilibrium under the current strategy set and adds the equilibrium strategy to the set. Policy-space response oracles (PSRO) [77] provides a unified perspective of FSP and DO, using a policy pool to train new policies to be added. In practice, when training RL algorithms, a model pool is usually maintained from which opponents are selected to collect data for training. Different protocols of opponent selection can be used, such as naive self-play that always chooses the latest model, delta-uniform self-play [78] that randomly chooses from recent models, population-based self-play [8,79] that creates different populations for weakness exploitation, and prioritized self-play [9] that chooses opponents according to winning rate.

Centralized Training Decentralized Execution (CTDE) is another popular paradigm in multi-agent learning that jointly trains multiple agents in a centralized manner but keeps their independence in execution. It is proposed to provide a mechanism of communication to eliminate the problems of unstable dynamics in independent training. Value-based algorithms like Value Decomposition Network (VDN) [27] and QMIX [28] are variants of DQN in cooperative multi-agent settings that adopts centralized state-action value functions, using summation and mixing network to combine individual Q-networks. QMIX also introduces the internal state of the environment in the mixing network. Multi-agent DDPG (MADDPG) [29] is a policy-based algorithm that generalizes DDPG to multi-agent settings. Each agent has its own actor and critic network and reward function so it can deal with both competitive and cooperative settings. The critics take observations and actions of other agents as input and are trained in a centralized manner. Counterfactual Multi-agent Policy Gradients (COMA) [30] extends vanilla actor-critic to the setting of Dec-POMDP where all agents share the same reward function. It uses a shared critic network with a counterfactual baseline function to assign credits to different agents.

4. Milestones of Game AI Systems

This section summarizes several important milestones of game AI in recent years. Each of these game AI systems uses different combinations of AI techniques listed in Section 3. The AI systems covered in this section are: AlphaGo series (AlphaGo [2], AlphaGo Zero [3], AlphaZero [4]) in the game of Go, HUNL AI systems (DeepStack [5], Libratus [6], Pluribus [7], AlphaHoldem [80]), Mahjong AI systems (Suphx [18], JueJong [19]), Doudizhu AI systems (DouZero [20], PerfectDou [21]), and AI systems for video games (AlphaStar [8], OpenAI Five [9], JueWu [10]).

4.1. Board Games

Go is a classic board game of much higher complexity than Chess. It is estimated that the game tree of Go has a branching factor of 250 and an average depth of 150, so the state-space complexity is up to 10^{360}. In 2016, Google DeepMind proposed AlphaGo [2], which beat professional human players using a combination of deep learning and tree search. In 2017, a new training framework was proposed to build AlphaGo Zero [3], which does not rely on human data and learns from scratch. The same framework is used to train AlphaZero [4], achieving superhuman performance in Go, Chess, and Shogi.

AlphaGo trains a policy network, a value network, and a rollout policy as prior knowledge. These components are combined with MCTS to select moves in real-time gameplay. The training pipeline consists of three phases. In the first phase, the policy network and the rollout policy are trained using 30 million state-action pairs from matches of professional humans. The policy network is a 13-layer convolutional neural network, while the rollout policy is based on a linear evaluation function of handcraft features, achieving lower accuracy but higher speed than the policy network. In the second phase, the policy network is improved by self-play reinforcement learning. It plays against previous versions of checkpoints and is optimized using policy gradient methods. The improved policy network can achieve an 80% winning rate against the original version. In the third phase, 30 million state-value pairs are collected from self-play matches of the policy network to form a high-quality dataset. A value network of 14 layers is trained on the dataset to learn an evaluation function to estimate the winning rate of board states.

Once the prior knowledge is trained, parallel MCTS is executed in real-time gameplay. The action probability produced by the policy network is used to guide the tree policy. Once reaching a leaf node, the fast rollout policy simulates until the end of the episode. The value of the leaf node is a linear combination of the final payoff and the evaluation of the value network. This scheme of using prior knowledge in MCTS is called APV-MCTS in their work. When running on clusters of 1202 CPUs and 176 GPUs with a thinking time of 5 s per move, AlphaGo achieves an Elo rating of 3140, reaching the level of professional play.

AlphaGo Zero uses a different framework to train a combined policy-value network to predict the actions and values of game states, which is a Resnet [81] that shares the first layers to extract features. A simpler variant of MCTS than AlphaGo is used, where the leaf node of the search tree is evaluated directly using the value network without rollout to the end of the game. The network is trained by reinforcement learning that uses MCTS as the policy improvement operator. Specifically, the training data is generated by the self-play of MCTS agents using the best historical checkpoint. The policy network is trained to approximate the action probability produced by MCTS, and the value network is trained to approximate the final payoff of the game. In other words, through self-play, the policy network is constantly approaching an improved MCTS policy in a supervised manner. After the network is trained on 29 million self-play games for over 40 days, it is combined with MCTS paralleled on 4 TPUs in real-time gameplay, achieving an Elo rating of 5185, much higher than that of the best human player, though it only takes three days of training to surpass old version of AlphaGo.

AlphaZero uses the same training framework as AlphaGo Zero but extends its application to other board games like Chess and Shogi. Data augmentation methods like rotation and reflection used in AlphaGo Zero are not adopted to accommodate a broader class of games. Besides, AlphaGo Zero uses the best checkpoint from all previous iterations to generate self-play data, while AlphaZero maintains the newest checkpoint and updates it continually. AlphaZero achieves state-of-the-art performance on all three games, with the training time of 9 h, 12 h, and 13 days on Chess, Shogi, and Go.

4.2. Card Games

4.2.1. HUNL

Heads-up No-limit Hold'em (HUNL) is a card game involving imperfect information and stochasticity. Two-player HUNL has more than 10^{160} information sets [82], making it impossible to solve by vanilla CFR methods. In 2017 Bowling and his team proposed DeepStack [5], which uses deep learning and continual re-solving to solve HUNL. Brown and his team built Libratus [6] in 2018 and Pluribus [7] in 2019 to solve two-player and six-player HUNL with a different method that combines nested-safe subgame solving with a blueprint strategy on an abstracted game. In 2022 Zhao and his team [80] applied deep reinforcement learning on HUNL, achieving similar results to previous methods.

DeepStack, Libratus, and Pluribus use the same real-time planning technique but give different names as continual re-solving and nested safe subgame solving. The basic idea is to decompose an imperfect information game into the trunk, the first few layers of the game tree, and subgames and solve each subgame when actually encountered in gameplay. By using players' ranges and opponent counterfactual values, as constraints, the re-solved subgame strategy is guaranteed to be not more exploitable when combined with the trunk strategy. The ranges can be updated by Bayes rules at each action and random event when playing, and the opponent counterfactual value is the prior knowledge to be trained and stored in models.

DeepStack trains deep counterfactual value (DCFV) networks to predict counterfactual values of both players given the ranges at each information set. The game of HUNL can be divided into four stages, preflop, flop, turn, and river, and DeepStack trains three DCFV networks at each of the first three stages. In this way, a search tree expanded at any information set can reach one network or the end of the game within a limited depth. By further limiting the action space, the size of such depth-limit sparse look-ahead trees can be brought down to around 10^7 to be solved in seconds. The DCFV networks are trained in reverse order by supervised learning on millions of data pairs generated by randomly initializing ranges at each stage and running continual re-solving with the network in the next stage. In this way, the prior knowledge of counterfactual value prediction in each stage of the game is stored in network models and is combined with continual re-solving in real-time gameplay, beating all 11 professional players in a four-week match.

Instead of dividing the game into stages, Libratus and Pluribus use abstraction to reduce the complexity of HUNL and calculate a blueprint strategy on the abstracted game using MCCFR. Both state abstraction and action abstraction are used to group similar card combinations and limit the choices of betting size. Such blueprint strategy and the subgame values in each abstracted information set are stored in tabular form.

When playing the game in real-time, Libratus always plays according to the blueprint strategy by rounding an off-tree opponent bet size in the first two betting rounds where the abstraction is dense, but will use nested safe subgame solving to re-solve the subgame strategy for off-tree opponent action in the last two betting rounds. To compensate for the error brought by action abstraction in early rounds, Libratus further includes a self-improvement module. Over the 20 days of Brains vs. AI challenge with four professional humans, Libratus analyzes the most frequent bet sizes used in the first betting round by humans and chooses three of them each day to add to and finetune the blueprint strategy overnight, making the abstracted betting sizes of the first round of the blueprint strategy denser. In this way, Libratus gradually fixes its potential weakness of abstraction by leveraging the humans' ability. The competition shows that Libratus can beat professional players by a huge margin of 147 mbb/h.

However, solving to the end of the game in 6-player HUNL is computationally infeasible. When solving the subgames using CFR algorithm, Pluribus assumes that each opponent can only choose among four strategies, the blueprint strategy and its shifted version towards folding, calling, and raising. In each state of the subgame, the opponents always take actions based on the selected strategy, instead of choosing from an abstracted action space in each state as in Libratus. Such an assumption dramatically reduces the size of subgames so that nested safe subgame solving can be performed. Monte Carlo Linear CFR solves large subgames, and vector-based Linear CFR solves small subgames for higher accuracy. Pluribus does not share the self-improvement module with Libratus, probably because of the higher training cost of the blueprint strategy than Libratus. Experiments show that Pluribus achieves superhuman performance when playing with professional humans and is evaluated under AIVAT to reduce variance.

AlphaHoldem uses deep reinforcement learning to train a policy-value network as prior knowledge and only performs one network forwarding at each state in real-time gameplay. The neural network receives a player's observation as input and predicts the action to choose and the estimated value of the state. The training of the network is based on trinal-clip PPO, a variant of the PPO algorithm that adds an upper clipping bound to the importance sampling ratio when the advantage is negative, and a reward clipping to reduce high variance introduced by the stochasticity of the game. During self-play training, the data is collected by having the main agent compete with K-best historical checkpoints to keep both the diversity and quality of the opponent pool. The training takes three days on a server with 8 GPUs and 64 CPUs, generating 6.5 billion samples (2.7 billion hands) of training data. Experiments show that AlphaHoldem performs at a similar level to DeepStack, but with a much faster inference time.

4.2.2. Mahjong

Mahjong is a multiplayer imperfect-information game that is challenging for AI research due to its complex scoring rule and rich hidden information. The average size of information sets is around 10^{48} in Riichi Mahjong and 10^{11} in 1-on-1 Mahjong, which is much larger than that of HUNL (about 10^3). Planning algorithms used in HUNL, like subgame re-solving, are not applicable because of longer game length and difficulty in state abstraction. In 2020 MSRA built Suphx [18] to solve Riichi Mahjong with deep reinforcement learning, outperforming most top human players. In 2022 Tencent AI Lab built JueJong [19] to solve 1-on-1 Mahjong with a combination of CFR and actor-critic framework, beating a human champion.

Suphx trains five policy networks as prior knowledge, which are embedded in a decision flow to be used in real-time gameplay. The training pipeline consists of two phases. In the first phase, five policy networks used in different decision points are trained, each using 4 to 15 million state-action pairs of top human players. All of the five networks use a similar structure to Resnet. In the second phase, only the policy network of the discard model is improved by a variant of the policy gradient algorithm that adds a term of entropy regularization and uses dynamic entropy weight to stabilize the training. The discard model is embedded in the decision flow and always plays against the latest model to collect training data. Since a game of Mahjong consists of multiple rounds, and the strategies vary in different rounds based on cumulative round scores, the reward of each round in RL training is decided by a global reward predictor, which is a recurrent network to predict game rewards after several rounds. This model is trained in advance on state-value pairs of human data using supervised learning. To speed up the training in the imperfect-information setting, Suphx uses a technique called oracle guiding that exposes hidden information as perfect features to the trained model while gradually dropping them out until the oracle agent transits to a normal agent.

In real-time gameplay, Suphx uses the decision flow with five policy networks to choose an action. However, because the initial hand of each round tends to cause different styles of policies, Suphx re-finetunes the policy model at the beginning of each round using parametric Monte-Carlo Policy Adaptation (pMCPA). Specifically, several different trajectories are generated by randomly sampling opponents' hands and running the policy models. The trajectories are used to perform gradient updates to finetune the policy model. Experiments show that Suphx surpasses 99.99% of players in Tenhou and achieves higher stable ranks than professional players.

JueJong trains a policy-value network as prior knowledge and only performs one network forwarding at each state in real-time gameplay. The training of the policy-value network is based on Actor-Critic Hedge (ACH), a practical actor-critic implementation of Neural Weighted CFR that proves to converge to a Nash equilibrium. Specifically, the value network is trained to approximate the final payoff of the game, while the policy network is trained to minimize cumulative counterfactual regrets instead of maximizing cumulative rewards as in typical RL algorithms. The regret is calculated based on the state-action values predicted by the value network under the current state and replaces the advantage function in traditional policy gradient methods to fit in a distributed actor-critic training framework. Only the latest model is used to produce training data in a self-play manner. Experiments show that JueJong is significantly more difficult to exploit than agents trained by other RL algorithms and achieves superhuman performance in head-to-head evaluation with human players.

4.2.3. Doudizhu

Doudizhu is a trendy three-player card game in China that involves imperfect information and cooperation. In the game, two peasants cooperate against one landlord and receive the same payoff. It is estimated that the average size of information sets in Doudizhu is around 10^{23} [83], and the hidden information makes it challenging to learn cooperative behavior. Besides, Doudizhu has a large discrete action space that cannot be easily abstracted. In 2021 DouZero [20] used deep reinforcement learning to solve Doudizhu from scratch, ranking first in the Botzone leaderboard [84] among 344 AI agents. In 2022 PerfectDou [21] proposed perfect information distillation and outperformed DouZero, with much higher training efficiency. Neither AI systems are evaluated by competing with top human players, so no superhuman performance is reported yet.

DouZero trains three value networks as prior knowledge to predict the state-action value for three positions. Since there is no policy network, it chooses the action with the highest value in real-time gameplay. The neural network is a fully connected network that receives the encoding of a state and an action and predicts the expected payoffs under the current state. The state does not include the hidden information of other players, and the reward is simply the final score of the game. The training uses Monte Carlo value targets instead of bootstrapping TD targets to speed up convergence, and the self-play of the latest model generates the training data. Trained from scratch with 4 GPUs and 48 CPUs, DouZero beats DeltaDou in 10 days, one of the best Doudizhu agents before DouZero.

PerfectDou trains a policy-value network as prior knowledge and only performs one network forwarding at each state in real-time gameplay. The training is based on distributed PPO, a variant of IMPALA that uses GAE value targets as in PPO instead of V-trace. The actor network only takes each player's imperfect observation as input, while the critic network has access to the hidden information of other players. PerfectDou calls this scheme perfect-training-imperfect-execution (PTIE), or perfect information distillation, which can reduce the variance caused by imperfect information. An oracle reward is designed as the relative speed to empty one's hand between peasants and the landlord, which is a dense signal and is added to speed up the training in the early phase. As in DouZero, the training data is generated by the self-play of the latest model. Experiments show that PerfectDou achieves not only better performance than DouZero, but also higher efficiency of 10 times fewer training samples. PerfectDou beats some skilled human players in their evaluation, but no professional players are invited to the experiment.

4.3. Video Games

StarCraft, Dota 2, and Honour of Kings (HoK) are multiplayer online video games that are popular and played by millions of people, with human competitions held each season. These games are very complex and difficult to solve by AI algorithms because of the huge state and action space, the imperfect information of game states, cooperation between agents, the balance between long-term targets and short-term benefits, and heterogeneous agents involving different policies. In 2019 DeepMind applies distributed deep reinforcement learning to StarCraft and proposes AlphaStar [8], beating professional human players. In the same year, OpenAI Five [9] beats OG, the world champion team in Dota 2, also based on distributed DRL with huge computational power. In 2020 Tencent AI Lab proposed JueWu [10], which achieves superhuman performance on HoK while not limiting the hero pool as OpenAI Five does.

All of these AI systems are trained under a distributed actor-critic training framework. In this framework, multiple actors distributed in different machines asynchronously interact with the environment to collect training data and send them to a centralized replay buffer, and a learner, which may have multiple GPUs and be distributed on multiple machines, samples data from the buffer to train the neural network. A model pool is also maintained to save historical checkpoints or different populations of agent to sample opponents for the training. Policy-value networks are trained as prior knowledge and are used to produce action in real-time gameplay without any planning algorithms.

The training pipeline of AlphaStar consists of two phases. In the first phase, supervised learning is used to train initial model parameters from a dataset consisting of 971,000 games played by the top 22% players. Since there are three races with different mechanisms in StarCraft, three distinct models are trained. In the second phase, these models are improved by distributed reinforcement learning, which combines multiple techniques of TD(λ) [85], V-trace [58] and upgoing policy update (UPGO) [8]. Specifically, the policy network is updated using the loss function of clipped importance sampling, while the value network is updated using TD(λ). UPGO is a new method proposed by AlphaStar which updates the policy from partial trajectories with better-than-expected returns by bootstrapping when the behavior policy takes a worse-than-average action. In this way, the policy always moves towards better trajectories.

To prevent models from sticking to the local optimal in self-play training, AlphaStar trains three kinds of populations of agents: the main agents, the main exploiters, and the league exploiters, each containing checkpoints of past versions. Specifically, the main agents train against all past agents as well as themselves and are constantly improved over time. The main exploiters only train against the main agents to exploit their weakness, and the league exploiters train against all past agents. Main exploiters and league exploiters are re-initialized when adding new checkpoints to the population. Using population-based self-play, AlphaStar overcomes the problem of non-transitivity in strategies, making the main agents hard to exploit.

OpenAI Five trains the policy-value network using distributed PPO. Instead of population-based self-play, OpenAI Five uses heuristic self-play where the main agent plays against the latest policy for 80% of games and the past checkpoints for 20% of games to generate training data. During the 10-month training, many restarts and reverts happen when the environment or the model's architecture changes. OpenAI Five proposes a collection of tools called continual transfer via surgery to avoid training the model from scratch at every restart. The basic idea is to ensure the new policy implements the same mapping from observation to action probability despite the changes in observation space or network structure so that the training can be smoothly restored.

In the game of Dota 2 and HoK, two teams select five heroes in turn from a hero pool before playing the game to battle. Such a drafting process can be considered a separate game to provide initial configurations of the main game. To prevent an exponential number of hero combinations, OpenAI Five limits the hero pool to 17 heroes (yielding around 4.9 million combinations) and uses randomly selected lineups in the training phase. The winning rate of a fixed lineup can be estimated using the predicted score at the first few frames of the game. During real-time game-play, a drafting program uses the minimax algorithm to pick the hero that maximizes the winning rate under the worst case of opponent hero selection. Further experiments show that including more heroes in training is likely to cause degraded performance and much slower training speed.

JueWu introduces the idea of curriculum learning to deal with exponential numbers of hero combinations and extends to a hero pool of 40 heroes. The training pipeline consists of three phases. In the first phase, several fixed lineups with a balanced winning rate are selected based on a vast amount of human data, and one teacher model is trained for each lineup. In the second phase, a student model is trained by supervised learning to learn the general behavior of those teacher models under different hero lineups. This process of knowledge distillation uses action probabilities as soft targets and adopts Shannon entropy as the loss function. The student model is further improved in the third phase by reinforcement learning under randomly selected lineups to generalize to arbitrary hero combinations. The RL algorithm used is dual-clip PPO which adds an upper clipping bound to the importance sampling ratio when the advantage is negative to reduce variance. JueWu uses the same heuristic self-play scheme with OpenAI Five, which plays against 80% of the latest and 20% of past models to prevent strategy collapse. In real-time gameplay, MCTS instead of minimax is used in drafting because a complete search tree is too large to explore with 40 heroes. A win-rate predictor is trained in advance by supervised learning on match datasets from self-play training. The winning rate of leaf nodes in MCTS is predicted by this small model rather than using the complete model as in OpenAI Five to speed up the inference.

5. Paradigms and Trends

Different AI systems designed for various games use diverse combinations of AI techniques, and it is difficult to find a universal pattern if we only study each in isolation. To thoroughly compare these AI systems and analyze the reasons for their success, this survey breaks each of them into basic components and categorizes these techniques based on the problems they tackle, which are directly related to the characteristics of the games. By comparing the basic techniques used in these AI systems, we discuss the following questions in this section: (1) Are there any paradigms capable of solving different types of games in these systems? (2) How do different game features affect the selection of techniques used to tackle them? (3) Which paradigm will become the general solution of game AI, and which games will likely be addressed in the future?

5.1. Common Paradigms

Table 2 shows an overview of the main components of each game AI system. These AI systems learn some prior knowledge in an offline training phase and use that knowledge in real-time inference. The prior knowledge is policy or value models stored in neural networks or tabular forms. The training usually involves reinforcement learning for policy/value improvement and supervised learning from human data as model initialization. When the environment model is known, the prior knowledge can be combined with real-time planning algorithms to conduct a tree search to produce better policies. In general, there are three kinds of paradigms used in these milestones.

5.1.1. AlphaGo Series

AlphaGo, AlphaGo Zero, and AlphaZero follow a common paradigm to solve classic board games, where the policy and value models are trained and combined with MCTS in real-time gameplay. While AlphaGo uses supervised learning and policy gradient to train the networks, AlphaGo Zero and AlphaZero adopt MCTS as policy improvement operators in the RL training. MCTS is heavily used both as planning and learning algorithms for two reasons. First, MCTS can be used here because these classic board games are all perfect-information games where the environment model is known, and a fast response of actions is not required. Second, board games like Go, Chess, and Shogi have such large state space that a perfect policy or value model is infeasible, but a relatively small time horizon that real-time planning algorithms like MCTS can improve the performance of an imperfect model by a large margin. The feasibility and superiority of the MCTS application are the keys to the success of the AlphaGo series in perfect-information games.

However, this paradigm is limited to perfect-information games because there are some limitations to the use of MCTS. First, an explicit environment model is required to predict the next state under current action. Second, the action space should be discrete and limited so that the branching factor of the search tree is under control. Third, MCTS only expands trees with limited depth and is inefficient to bootstrap values from the end of the episode to the early stages of the game when the time horizon is long. Though MuZero [86] overcomes the first limitation by training an additional representative model to simulate the environment and achieves success on Atari 2600 video games, the latter two limitations make it unable to apply to games with imperfect information or long time horizons.

Table 2. The main components of AI system milestones.

AI System	Prior Knowledge	Training Pipeline	Inference	RL Algorithm
AlphaGo	policy network rollout policy value network	SL + RL SL SL	MCTS + NN	PG
AlphaGo Zero	policy-value network	RL	MCTS + NN	MCTS-RL
AlphaZero	policy-value network	RL	MCTS + NN	MCTS-RL
DeepStack	DCFV network	SL	Continual re-solving + NN	N/A
Libratus	blueprint strategy	Abstraction + MCCFR	Nested-safe subgame solving	N/A
Pluribus	blueprint strategy	Abstraction + MCCFR	Nested-safe subgame solving	N/A
AlphaHoldem	policy-value network	RL	NN	Trinal-clip PPO
Suphx	policy networks global reward predictor	SL + RL SL	pMCPA finetune + NN	PG with entropy
JueJong	policy-value network	RL	NN	ACH
DouZero	value network	RL	One-step greedy + NN	DMC
PerfectDou	policy-value network	RL	NN	PPO
AlphaStar	policy-value network	SL+RL	NN	UPGO
OpenAI Five	policy-value network	RL	Minimax drafting NN	PPO
JueWu	policy-value network drafting value network	RL+SL+RL SL	MCTS drafting NN	Dual-clip PPO

5.1.2. CFR Series

DeepStack, Libratus, and Pluribus follow a unique paradigm to solve HUNL by CFR algorithms. The core of these systems is to simplify the original game by abstraction and use nested safe subgame solving for real-time planning. Specifically, DeepStack decomposes HUNL into stages and trains value networks at each stage, which is combined with depth-limit sparse look-ahead trees to re-solve subgame policies. In this way, DeepStack always calculates real-time responses to opponent off-tree actions but suffers from approximation error of the value networks. Libratus and Pluribus directly apply abstraction to the whole game and compute a tabular blueprint strategy and the corresponding counterfactual values. Subgame policies are later re-solved based on the value of blueprint strategies. Such abstraction can introduce error when rounding off-tree betting size to the course-grained blueprint strategy but is more robust when the opponent mistakenly chooses an action far from optimal, which could cause unprecedented ranges and value estimation in DeepStack.

However, this paradigm is only a success limited to HUNL, and no further works successfully apply it to larger imperfect-information games. The reason lies in the compromise of computational cost and approximation error. Specifically, the computational cost under this paradigm comes from the CFR iterations both in the training and inference stages, and experiments show that a large number of CFR iterations are needed to produce a high-quality solution [5]. The blueprint strategies are in tabular forms, which are also limited by the available memory resources. Meanwhile, the process of subgame solving expands a search tree of limited depth and branching factor, so more abstraction is needed in larger games. Pluribus uses a very strong assumption that each player can only choose among four strategies to bring down the size of search trees, which introduces large errors in finding optimal policies. In fact, HUNL is relatively small in imperfect-information games considering its average size of information sets and the short time horizon. It can be concluded that the CFR-based paradigm is not scalable to larger games. It achieves superhuman performance on HUNL mainly because of the limited complexity of the game and suboptimality of human strategies.

5.1.3. DRL Series

All other AI systems in Table 2 follow a common paradigm of distributed deep reinforcement learning. Policy-value networks are trained as prior knowledge and directly used to produce actions in real-time inference without any planning algorithm. This paradigm is more general as it does not require a model of the environment. The networks are trained under a distributed actor-critic framework using algorithms like PPO or its variants to be easily scaled to an arbitrary amount of computational resources. Suphx and AlphaStar also use supervised learning to train initial models from human data to speed up the RL training. The training data is generated by playing the main agent against opponents sampled from a model pool to enrich policy diversity and prevent strategy collapse.

Though these AI systems all follow the same paradigm, there are some differences in the choices of each component related to the characteristics of the specific games they tackle. For example, in Riichi Mahjong, one full match consists of several rounds, and the strategies vary in each round, which is different from other games where each match is independent and does not affect the other. To deal with this issue, Suphx [18] trains an extra recurrent network as a global reward predictor to shape the reward of each match. AlphaHoldem [80] suffers from the large variance introduced by the stochasticity of HUNL and uses a variant of PPO with additional clipping to stabilize the training process. JueJong [19] seeks to find a policy with lower exploitability to approximate the Nash equilibrium, so the CFR-based ACH algorithm is used as the RL algorithm instead of PPO to minimize the cumulative regrets of the trained strategy. JueWu [10] deals with a large hero pool with so many hero combinations that a drafting value network is needed to assist the MCTS for fast value prediction.

5.2. Techniques for Game Features

In addition to common paradigms, we also notice that different techniques are selected for some common game features in these milestones. As is shown in Table 3, we make a detailed comparison of these AI systems from the following perspective: the self-play scheme used in multi-agent settings, the methods to deal with imperfect information, and the algorithms to learn policies for heterogeneous agents.

Table 3. The selected techniques for common game features in AI system milestones.

AI System	Self-Play Scheme	Imperfect Information	Heterogeneous Agents
AlphaGo	Uniform Random	N/A	N/A
AlphaGo Zero	Best		
AlphaZero	Latest		
DeepStack	N/A	CFR	
Libratus			
Pluribus			
AlphaHoldem	K-Best	No use	
Suphx	Latest	Oracle Guiding	
JueJong	Latest	ACH	
DouZero	Latest	No use	
PerfectDou	Latest	PID	
AlphaStar	Population	PID	Population
OpenAI Five	Heuristic	No use	Random
JueWu	Heuristic	PID	Knowledge Distillation

5.2.1. Self-Play Scheme

In multi-agent settings, directly applying single-agent RL algorithms to train agents independently is not guaranteed to converge because most games exhibit non-transitive behavior. For example, in the Rock-Paper-Scissor game, rock beats scissor and scissor beats paper, but rock could not beat paper, forming a cyclic sub-structure of the policy space. Previous works suggest that real-world games look like spinning tops [87], where relatively weak strategies tend to form longer circles. So a population of agents is necessary for the self-play in multi-agent training to learn policies with lower exploitability. In practice, several self-play schemes are used in these AI systems.

AlphaZero, Suphx, JueJong, DouZero, and PerfectDou use naive self-play, the simplest form of self-play that only chooses the latest model as the opponent. AlphaGo randomly samples opponents from historical checkpoints for more diversity. OpenAI Five and JueWu use a heuristic scheme that samples 80% of the latest model and 20% of historical models to put more weight on the latest model. AlphaGo Zero uses the best of the historical models for self-play to generate training data of high quality. AlphaHoldem chooses the K-best models to keep both the diversity and quality of the opponent pool. Most of the self-play schemes chosen in these milestones are for no specific reasons, except for PerfectDou, which proves by experiments that K-best self-play performs better than other self-play techniques.

5.2.2. Imperfect Information

Learning a Nash equilibrium in imperfect-information games is more difficult because the hidden information of other players is not included in the current observation and has to be inferred from other players' past actions. CFR-based algorithms generally consider the game as an indivisible whole and minimize the cumulative regrets at each information set to approach a Nash equilibrium. DeepStack, Libratus, and Pluribus combine variants of CFR algorithms with decomposition and abstraction to handle HUNL, which has too many information sets to be solved by vanilla CFR. JueJong proposed Actor-Critic Hedge to solve 1-on-1 Mahjong in a distributed RL framework, which is also based on CFR.

However, RL algorithms learn policies as mappings from observations to actions by interacting with the environment and optimizing the expected cumulative rewards, which can be of high variance in imperfect information settings. Two methods are proposed to reduce the variance and stabilize the training. Suphx uses oracle guiding to handle imperfect information by first exposing the hidden information to the policy and value network to train an oracle agent while gradually dropping them out until the oracle agent transits to a normal agent. However, this way of continual training is still unstable and requires additional tricks to converge [18]. PerfectDou, AlphaStar, and JueWu use perfect information distillation (PID), which exposes the hidden information to the centralized value network to reduce variance, while the policy network does not rely on hidden information in real-time inference. Theoretical analysis shows that PID is a more natural way to handle imperfect information and can generalize to larger games [88].

5.2.3. Heterogeneous Agents

MOBA games are usually designed to have heterogeneous agents with quite different mechanics and strategies. Such games provide another challenge for AI research since an AI system has to train different models for each kind of agent or a unified model which can generalize to distinct policies under different settings. There are three races of agents in StarCraft with different mechanics, so AlphaStar trains separate models for them by creating distinct populations of main agents, main exploiters, and league exploiters for each race. However, in games like Dota 2 and HoK, the number of heterogeneous agents called heroes is large, and there is an exponential number of hero combinations in the 5-versus-5 setting. OpenAI Five trains a unified model to control different heroes by limiting its application to 17 heroes and randomly sampling hero combinations for each game during the training. It suffers from a slow convergence speed and is hard to extend to more heroes, as shown in their further experiments with 25 heroes.

JueWu proposes a new training paradigm under the setting of MOBA games by noticing that many heroes can be classified into several positions with similar strategies for each position, and most of the hero combinations are improper and not preferred by human players. JueWu chooses some typical hero combinations based on human data and trains a separate teacher model for each combination. Knowledge distillation is then used to train a student model to learn the general strategy under different hero combinations by imitating the behavior of teacher models. Such a student model is further trained under random hero combinations to generalize to arbitrary settings. This idea of curriculum learning that learns general behavior from specific settings is the key to the scalability of JueWu, which achieves superhuman performance with a hero pool of 40 heroes.

5.3. Future Trends

As Sutton said [89], "The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective". So what is the general paradigm in the field of game AI? As is discussed in this section, there are three kinds of paradigms used in these milestones. AlphaGo series and CFR series take advantage of tree search, either MCTS or nested safe subgame solving, and cannot scale to larger games with continuous action space or long time horizons. DRL series train models by advanced self-play with distributed deep reinforcement learning, which is the most general and promising paradigm for three reasons. First, deep neural networks can approximate arbitrary mappings from observations to actions as the policy model and are not restricted to games with finite state or action space. Second, advanced self-play schemes can be adopted to approach a Nash equilibrium with minimal exploitability under multi-agent settings by creating populations of diverse agents as opponents. Third, distributed asynchronous actor-critic training framework like IMPALA is easily scaled to an arbitrary amount of computational resources so that this paradigm is highly scalable to games of higher complexity once more computational resources are available, which is guaranteed by Moore's law [90], or its generalized version of continued exponentially falling of computational cost.

As researchers turn to games with higher complexity, we believe that real-world games, especially sports games, will become the next popular AI benchmarks because of their higher complexity and the inspirations that AI strategies can bring to humans. For example, Gran Turismo is a racing game that precisely reproduces the non-linear control challenges of real race cars, in which a superhuman AI trained by DRL [91] gives inspiration to a professional player and improve his performance. Another example is Google Research Football Environment [92], a football simulator to provide a real-world multiplayer sports game as a challenge for AI research. Previous works [93] have shown that with enough computational power, agents trained by DRL can create a self-supervised auto-curriculum and learn complex emergent behaviors of human-relevant skills without any human guidance. By applying the DRL-based training paradigm to real-world games such as football, it is expected that human-relevant skills can be learned from scratch and inspire humans with new tactics or even innovate the field when AI surpasses human performance.

6. Conclusions

Though AI systems have reached superhuman performance in many complex games in recent years, diverse techniques are used in each of them, and it is hard to see both the key to their success and their limitations. This survey aims to give a detailed analysis of the techniques and paradigms used in modern game AI systems in light of game features. We first summarize the common features in various games and show their challenges to AI research. By systematically reviewing the AI techniques typically used to create game-playing agents, we find that in different times of game AI history, the invention of new algorithms is mainly driven by the increasing complexity of new game features, and many of these algorithms are designed to tackle specific features. Since modern game AI systems are composed of multiple techniques, we propose a novel framework to break these systems into basic algorithmic components and compare them based on the game features. By analyzing the choices of these components and the game features, we extract three common paradigms to build AI systems for different games. Based on the mechanics of these paradigms and the features in modern games, we conclude that deep reinforcement learning is the most general and scalable paradigm to train strong AIs in games with higher complexity. We hope this survey can provide a comprehensive review of modern game AI systems and the basic techniques involved, to inspire researchers to build AI systems for larger games, such as real-world sports games.

Author Contributions: Writing—original draft preparation, Y.L.; Writing—review and editing, W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Science and Technology Innovation 2030—"The New 34 Generation of Artificial Intelligence" Major Project No.2018AAA0100901, China, and by Project No.2020BD003 supported by PKU-Baidu Fund.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Turing, A.M. Computing machinery and intelligence. In *Parsing the Turing Test*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 23–65.
2. 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]
3. 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]
4. 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]
5. Moravčík, M.; Schmid, M.; Burch, N.; Lisý, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; Bowling, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* **2017**, *356*, 508–513. [CrossRef]
6. Brown, N.; Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* **2018**, *359*, 418–424. [CrossRef]
7. Brown, N.; Sandholm, T. Superhuman AI for multiplayer poker. *Science* **2019**, *365*, 885–890. [CrossRef]
8. Vinyals, O.; Babuschkin, I.; Chung, J.; Mathieu, M.; Jaderberg, M.; Czarnecki, W.M.; Dudzik, A.; Huang, A.; Georgiev, P.; Powell, R.; et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind Blog* **2019**, *2*. Available online: https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii (accessed on 8 August 2022).
9. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, 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.
10. Ye, D.; Chen, G.; Zhang, W.; Chen, S.; Yuan, B.; Liu, B.; Chen, J.; Liu, Z.; Qiu, F.; Yu, H.; et al. Towards playing full moba games with deep reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 621–632.
11. Risi, S.; Preuss, M. From chess and atari to starcraft and beyond: How game ai is driving the world of ai. *KI-Künstliche Intell.* **2020**, *34*, 7–17. [CrossRef]
12. Yin, Q.; Yang, J.; Ni, W.; Liang, B.; Huang, K. AI in Games: Techniques, Challenges and Opportunities. *arXiv* **2021**, arXiv:2111.07631.
13. Copeland, B.J. The Modern History of Computing. Available online: https://plato.stanford.edu/entries/computing-history/ (accessed on 10 August 2022).
14. Tesauro, G. Temporal difference learning and TD-Gammon. *Commun. ACM* **1995**, *38*, 58–68. [CrossRef]
15. Schaeffer, J.; Lake, R.; Lu, P.; Bryant, M. Chinook the world man-machine checkers champion. *AI Mag.* **1996**, *17*, 21–21.
16. Campbell, M.; Hoane, A.J., Jr.; Hsu, F.h. Deep blue. *Artif. Intell.* **2002**, *134*, 57–83. [CrossRef]
17. Bowling, M.; Burch, N.; Johanson, M.; Tammelin, O. Heads-up limit hold'em poker is solved. *Science* **2015**, *347*, 145–149. [CrossRef]

18. Li, J.; Koyamada, S.; Ye, Q.; Liu, G.; Wang, C.; Yang, R.; Zhao, L.; Qin, T.; Liu, T.Y.; Hon, H.W. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv* **2020**, arXiv:2003.13590.
19. Fu, H.; Liu, W.; Wu, S.; Wang, Y.; Yang, T.; Li, K.; Xing, J.; Li, B.; Ma, B.; Fu, Q.; et al. Actor-Critic Policy Optimization in a Large-Scale Imperfect-Information Game. In Proceedings of the International Conference on Learning Representations, Virtual Event, 3–7 May 2021.
20. Zha, D.; Xie, J.; Ma, W.; Zhang, S.; Lian, X.; Hu, X.; Liu, J. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, Virtual Event, 18–24 July 2021; pp. 12333–12344.
21. Guan, Y.; Liu, M.; Hong, W.; Zhang, W.; Fang, F.; Zeng, G.; Lin, Y. PerfectDou: Dominating DouDizhu with Perfect Information Distillation. *arXiv* **2022**, arXiv:2203.16406.
22. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
23. Schwalbe, U.; Walker, P. Zermelo and the early history of game theory. *Games Econ. Behav.* **2001**, *34*, 123–137. [CrossRef]
24. Osborne, M.J.; Rubinstein, A. *A Course in Game Theory*; MIT Press: Cambridge, MA, USA, 1994.
25. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
26. Watson, J. *Strategy: An Introduction to Game Theory*; WW Norton: New York, NY, USA, 2002; Volume 139.
27. Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W.M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J.Z.; Tuyls, K.; et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv* **2017**, arXiv:1706.05296.
28. Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; Whiteson, S. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4295–4304.
29. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6382–6393.
30. Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; Whiteson, S. Counterfactual multi-agent policy gradients. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32. .
31. Hart, P.; Nilsson, N.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
32. Stockman, G. A minimax algorithm better than alpha-beta? *Artif. Intell.* **1979**, *12*, 179–196. [CrossRef]
33. Kocsis, L.; Szepesvári, C. Bandit based monte-carlo planning. In Proceedings of the European Conference on Machine Learning, Berlin, Germany, 18–22 September 2006; pp. 282–293.
34. 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.
35. Gelly, S.; Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.* **2011**, *175*, 1856–1875. [CrossRef]
36. Chaslot, G.M.B.; Winands, M.H.; Herik, H. Parallel monte-carlo tree search. In Proceedings of the International Conference on Computers and Games, Beijing, China, 29 September–1 October 2008; pp. 60–71.
37. Ginsberg, M.L. GIB: Imperfect information in a computationally challenging game. *J. Artif. Intell. Res.* **2001**, *14*, 303–358. [CrossRef]
38. Bjarnason, R.; Fern, A.; Tadepalli, P. Lower bounding Klondike solitaire with Monte-Carlo planning. In Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling, Thessaloniki, Greece, 19–23 September 2009.
39. Frank, I.; Basin, D. Search in games with incomplete information: A case study using bridge card play. *Artif. Intell.* **1998**, *100*, 87–123. [CrossRef]
40. Cowling, P.I.; Powley, E.J.; Whitehouse, D. Information set monte carlo tree search. *IEEE Trans. Comput. Intell. Games* **2012**, *4*, 120–143. [CrossRef]
41. Whitehouse, D.; Powley, E.J.; Cowling, P.I. Determinization and information set Monte Carlo tree search for the card game Dou Di Zhu. In Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG'11), Seoul, Korea, 31 August 2011–3 September 2011; pp. 87–94.
42. Burch, N. Time and Space: Why Imperfect Information Games Are Hard. Available online: https://era.library.ualberta.ca/items/db44409f-b373-427d-be83-cace67d33c41 (accessed on 10 August 2022).
43. Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computing*; Springer: Berlin/Heidelberg, Germany, 2003; Volume 53.
44. Rechenberg, I. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*; Springer: Berlin/Heidelberg, Germany, 1978; pp. 83–114.
45. Dawkins, R.; Krebs, J.R. Arms races between and within species. *Proc. R. Soc. Lond. Ser. B Biol. Sci.* **1979**, *205*, 489–511.
46. Angeline, P.; Pollack J. Competitive Environments Evolve Better Solutions for Complex Tasks. In Proceedings of the 5th International Conference on Genetic Algorithms, San Francisco, CA, USA, 1 June 1993; pp. 264–270.
47. Reynolds, C.W. Competition, coevolution and the game of tag. In Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Boston, Massachusetts, USA, 6–8. July 1994; pp. 59–69.
48. Sims, K. Evolving 3D morphology and behavior by competition. *Artif. Life* **1994**, *1*, 353–372. [CrossRef]
49. Smith, G.; Avery, P.; Houmanfar, R.; Louis, S. Using co-evolved rts opponents to teach spatial tactics. In Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, Copenhagen, Denmark, 18–21 August 2010; pp. 146–153.

50. Fernández-Ares, A.; García-Sánchez, P.; Mora, A.M.; Castillo, P.A.; Merelo, J. There can be only one: Evolving RTS bots via joust selection. In Proceedings of the European Conference on the Applications of Evolutionary Computation, Porto, Portugal, 30 March–1 April 2016; pp. 541–557.
51. García-Sánchez, P.; Tonda, A.; Fernández-Leiva, A.J.; Cotta, C. Optimizing hearthstone agents using an evolutionary algorithm. *Knowl.-Based Syst.* **2020**, *188*, 105032. [CrossRef]
52. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
53. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
54. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
55. Konda, V.; Tsitsiklis, J. Actor-critic algorithms. In Proceedings of the Advances in Neural Information Processing Systems 12 (NIPS 1999), Denver, CO, USA, 29 November–4 December 1999; Volume 12.
56. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
57. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 20–22 June 2016; pp. 1928–1937.
58. Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 1407–1416.
59. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 7–9 July 2015; pp. 1889–1897.
60. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
61. Hart, S.; Mas-Colell, A. A simple adaptive procedure leading to correlated equilibrium. *Econometrica* **2000**, *68*, 1127–1150. [CrossRef]
62. Zinkevich, M.; Johanson, M.; Bowling, M.; Piccione, C. Regret minimization in games with incomplete information. *Adv. Neural Inf. Process. Syst.* **2007**, *20*, 1729–1736.
63. Tammelin, O. Solving large imperfect information games using CFR+. *arXiv* **2014**, arXiv:1407.5042.
64. Brown, N.; Sandholm, T. Solving imperfect-information games via discounted regret minimization. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 1829–1836.
65. Lanctot, M.; Waugh, K.; Zinkevich, M.; Bowling, M.H. Monte Carlo Sampling for Regret Minimization in Extensive Games. In Proceedings of the NIPS, Vancouver, BC, Canada, 6–11 December 2009; pp. 1078–1086.
66. Schmid, M.; Burch, N.; Lanctot, M.; Moravcik, M.; Kadlec, R.; Bowling, M. Variance reduction in monte carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 2157–2164.
67. Waugh, K.; Schnizlein, D.; Bowling, M.H.; Szafron, D. Abstraction pathologies in extensive games. In Proceedings of the AAMAS, Budapest, Hungary, 10–15 May 2009; pp. 781–788.
68. Waugh, K.; Morrill, D.; Bagnell, J.A.; Bowling, M. Solving games with functional regret estimation. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015.
69. Brown, N.; Lerer, A.; Gross, S.; Sandholm, T. Deep counterfactual regret minimization. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 793–802.
70. Li, H.; Hu, K.; Ge, Z.; Jiang, T.; Qi, Y.; Song, L. Double neural counterfactual regret minimization. *arXiv* **2018**, arXiv:1812.10607.
71. Steinberger, E. Single deep counterfactual regret minimization. *arXiv* **2019**, arXiv:1901.07621.
72. Steinberger, E.; Lerer, A.; Brown, N. DREAM: Deep regret minimization with advantage baselines and model-free learning. *arXiv* **2020**, arXiv:2006.10410.
73. Brown, G.W. Iterative solution of games by fictitious play. *Act. Anal. Prod. Alloc.* **1951**, *13*, 374–376.
74. Heinrich, J.; Lanctot, M.; Silver, D. Fictitious self-play in extensive-form games. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 7–9 July 2015; pp. 805–813.
75. Heinrich, J.; Silver, D. Deep reinforcement learning from self-play in imperfect-information games. *arXiv* **2016**, arXiv:1603.01121.
76. McMahan, H.B.; Gordon, G.J.; Blum, A. Planning in the presence of cost functions controlled by an adversary. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), Washington, DC, USA, 21–24 August 2003; pp. 536–543.
77. Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; Graepel, T. A unified game-theoretic approach to multiagent reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 4193–4206.
78. Bansal, T.; Pachocki, J.; Sidor, S.; Sutskever, I.; Mordatch, I. Emergent complexity via multi-agent competition. *arXiv* **2017**, arXiv:1710.03748.
79. Jaderberg, M.; Dalibard, V.; Osindero, S.; Czarnecki, W.M.; Donahue, J.; Razavi, A.; Vinyals, O.; Green, T.; Dunning, I.; Simonyan, K.; et al. Population based training of neural networks. *arXiv* **2017**, arXiv:1711.09846.

80. Zhao, E.; Yan, R.; Li, J.; Li, K.; Xing, J. AlphaHoldem: High-Performance Artificial Intelligence for Heads-Up No-Limit Texas Hold'em from End-to-End Reinforcement Learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual Event, 22 February–1 March 2022; Volume 36, pp. 4689–4697.
81. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
82. Johanson, M. Measuring the size of large no-limit poker games. *arXiv* **2013**, arXiv:1302.7008.
83. Zha, D.; Lai, K.H.; Cao, Y.; Huang, S.; Wei, R.; Guo, J.; Hu, X. Rlcard: A toolkit for reinforcement learning in card games. *arXiv* **2019**, arXiv:1910.04376.
84. Zhou, H.; Zhang, H.; Zhou, Y.; Wang, X.; Li, W. Botzone: An online multi-agent competitive platform for ai education. In Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, Larnaca, Cyprus, 2–4 July 2018; pp. 33–38.
85. Sutton, R.S. Learning to predict by the methods of temporal differences. *Mach. Learn.* **1988**, *3*, 9–44. [CrossRef]
86. 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. *Nature* **2020**, *588*, 604–609. [CrossRef]
87. Czarnecki, W.M.; Gidel, G.; Tracey, B.; Tuyls, K.; Omidshafiei, S.; Balduzzi, D.; Jaderberg, M. Real world games look like spinning tops. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17443–17454.
88. Lyu, X.; Baisero, A.; Xiao, Y.; Amato, C. A Deeper Understanding of State-Based Critics in Multi-Agent Reinforcement Learning. *arXiv* **2022**, arXiv:2201.01221.
89. Sutton, R. The bitter lesson. *Incomplete Ideas* **2019**, *13*, 12.
90. Schaller, R.R. Moore's law: Past, present and future. *IEEE Spectr.* **1997**, *34*, 52–59. [CrossRef]
91. Wurman, P.R.; Barrett, S.; Kawamoto, K.; MacGlashan, J.; Subramanian, K.; Walsh, T.J.; Capobianco, R.; Devlic, A.; Eckert, F.; Fuchs, F.; et al. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature* **2022**, *602*, 223–228. [CrossRef]
92. Kurach, K.; Raichuk, A.; Stańczyk, P.; Zając, M.; Bachem, O.; Espeholt, L.; Riquelme, C.; Vincent, D.; Michalski, M.; Bousquet, O.; et al. Google research football: A novel reinforcement learning environment. *arXiv* **2019**, arXiv:1907.11180.
93. Baker, B.; Kanitscheider, I.; Markov, T.; Wu, Y.; Powell, G.; McGrew, B.; Mordatch, I. Emergent tool use from multi-agent autocurricula. *arXiv* **2019**, arXiv:1909.07528.