Combining Code Generating Large Language Models and Self-Play to Iteratively Refine Strategies in Games

Yoram Bachrach[1] , **Edan Toledo**[1] , **Karen Hambardzumyan**[1] , **Despoina Magka**[1] , **Martin Josifoski**[1] , **Minqi Jiang**[1] , **Jakob Foerster**[1] , **Roberta Raileanu**[1] , **Tatiana Shavrina**[1] , **Nicola Cancedda**[1] , **Avraham Ruderman**[1] , **Katie Millican**[1] , **Andrei Lupu**[1] , **Rishi Hazra**[2]

[1] Meta, London, UK
[2] Örebro University, Sweden

Abstract

We propose a self-play approach to generating strategies for playing in multi-player games, where strategies are represented as computer code. We use large language models (LLMs) to generate pieces of code to play in the game, which we refer to as generated bots. We engage the LLM generated bots in competitions, designed to generate increasingly stronger strategies. We follow game theoretic principles in organizing these tournaments, and use a Policy Space Response Oracle (PSRO) approach. We start with an initial set of LLM generated bots, and continue in rounds for adding new bots into the population. Each round adds a bot to the population by asking the LLM to produce code for playing against a bot representing the Nash equilibrium mixture over the current population. Our analysis shows that even a few rounds are sufficient to produces strong bots for playing the game. Our demo shows the process for the game of Checkers. We allow users to select initial bots in the population, run the process, inspect how the bots evolve over time, and play against the generated bots.

1 Introduction

The recent innovation wave on Large Language Models (LLMs) has uncovered exciting applications, from translation to code generation [Li *et al.*, 2022; Fan *et al.*, 2023; Liu *et al.*, 2024a; Hou *et al.*, 2024]. We focus on leveraging LLM code generation for playing in multi-player games. Traditional game AI techniques have achieved remarkable successes in many games, but many computationally intensive methods such as Minimax search algorithms [Allis, 1994], Monte Carlo Tree Search (MCTS) [Browne *et al.*, 2012], or reinforcement learning (RL) [Silver *et al.*, 2018; Zhang *et al.*, 2021; Perolat *et al.*, 2022] result in game playing policies that are not interpretable [Angelov *et al.*, 2021]. LLMs offer a transformative approach through their ability to reason about strategy and generate code.

Our contribution: We propose a method for iteratively improving game-playing strategies using LLMs within a self-play framework guided by game theory. We use the framework of Policy Space Response Oracles (PSRO) [Lanctot *et al.*, 2017; Zhang *et al.*, 2024a], which iteratively adds strategies to a population by responding to the Nash equilibrium over the current strategies. Our key innovation lies in using a self-play approach that considers *strategies expressed in computer code*. We use an LLM to generate candidate pieces of code, and engage these bots in *tournaments* where they play the game one against the other. The LLM prompt asks for it to generate code for playing the game so as to maximize its win-rate against an opponent whose code is given in the prompt and reflects the current population of opponents. Thus, we use the LLM to generate increasingly more sophisticated strategies. Figure 1 shows an overview of our approach.

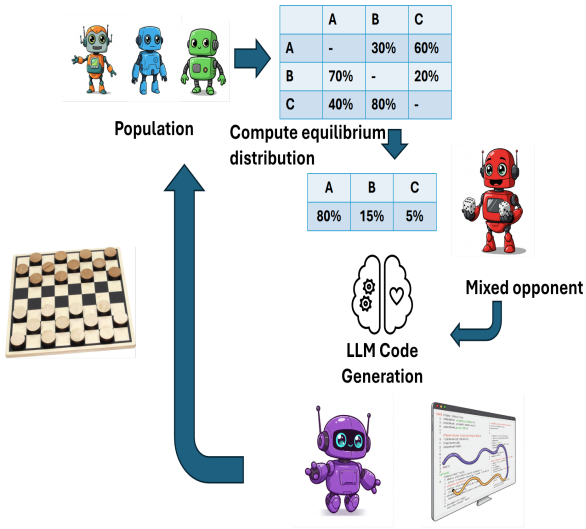

Figure 1: An overview of our approach, LLM-PSRO. We begin with a population of bots for playing the game (hand-written or LLM generated). In each round we: a. approximate the win-rate of each bot in the population against each other over many games and construct a payoff matrix with these win-rates; b. Approximate the Nash equilibrium in the game expressed as the payoff matrix; c. Generate code for the round's Mixture Bot, which calls each underlying bot with its probability under the Nash equilibrium. d. Get response candidates for the Mixture Bot by asking the LLM to generate code for playing in the game so as to best win against the Mixture bot's code; e. Select the best-response candidate, with the best win-rate against the Mixture Bot, and add it to the bot population.

Although the approach is general and can be applied to any multi-player game, we demonstrate it on the game of Checkers. Checkers possesses strategic depth and requires careful planning and adaptation [Griffith, 1974; Schaeffer *et al.*, 2007; Mańdziuk *et al.*, 2007; Knauer *et al.*, 2020], making it a challenging testbed that allows us to rigorously evaluate the effectiveness of our approach in LLM-driven gameplay.

2 Methods

We now describe our approach, using Checkers as an example. To iteratively improve gameplay, we grow a population of a bots for playing the game. Here, each bot is represented as the source code in Python for playing Checkers (based on a shared Python Checkers game engine).

Initialization We start with an initial population of bots $P_0 = (b_1, \ldots, b_h)$. We can either use hand-crafted bots, or query the LLM to generate a bot (code) for playing the game. We then **proceed in multiple rounds** of generating a new bot to add to the population. Each round r begins with **payoff table construction**, where for the current population of bots, we play all possible pairwise match-ups (bot b_i vs. bot b_j). We estimate the expected score $s_{i,j}$ of bot b_i playing against bot b_j by running $k = 1000$ games between them and taking the mean score (in Checkers, -1 for a loss, 0 for a tie, and $+1$ for a win). This data forms a payoff matrix P^r, where entry $P_{i,j}$ is the estimate for $s_{i,j}$. The payoff table captures the relative strengths of the bots in the current population.

Next we **approximate the Nash equilibrium distribution** over the bots, by taking the the payoff table and applying the fictitious play (FP) [Brown, 1951] method. FP iteratively simulates bots playing against the average strategy of their opponents. This process converges to the Nash Equilibrium distribution, which represents the optimal mixed strategy for the current population. Hence the Nash approximation takes in the payoff matrix P^r and returns an optimal mixture over the current bots — a probability x_i^r for each bot b_i (where $\sum_i x_i^r = 1$). We use the weights x_i^r to **construct the code for the Mixture Bot** b^r for the round. The Mixture Bot code does not employ a single fixed strategy, but rather before playing each game it samples a bot b_i from the current population according to the probability distribution $(x_1^r, x_2^r, \ldots, x_d^r)$, where d is the size of the bot population in round r; it then uses the sampled bot's strategy to decide on moves in the game. The Mixture Bot embodies the optimal mixed strategy derived from the Nash Equilibrium.

We use the Mixture Bot for **generating a new bot to add to the population**. We generate a new bot by prompting the LLM to examine the code of the Mixture Bot (given in the prompt), and to generate a new bot code so as to best defeat the Mixture Bot. **Our LLM prompt** asks the LLM to generate a best response to the current Mixture Bot, and includes the full code of the Mixture Bot (including the code for each of the sub-strategies it employs, i.e. the previous generation bots). As generating a best response bot is a challenging task, we repeat the LLM query t times, each time generating a new candidate bot, $c_1, \ldots, c_t$. The win-rate of each of these against the Mixture Bot b^r is estimated over $k = 1000$ games, and the best new bot is selected and added to the population.

By applying the above process m times, we obtain m additional bots in the population (and m Mixture bots). These are stronger and stronger bots, as each of these is designed to perform well against all the previous generations. Hence the LLM queries generate increasingly more effective strategies in response to the evolving mixture bot, so the process gradually improves the overall playing strength. Our full algorithm, called LLM-PSRO, is given in Algorithm 1.

Algorithm 1 LLM-PSRO: LLM-based Population Self-play for Game Playing Strategy Improvement

Require: Game simulator (e.g., Checkers engine)
Require: LLM for code generation
Require: Initial population of bots $P_0 = (b_1, \ldots, b_h)$
for $r = 1$ to m **do**
  for $i = 1$ to $|P_{r-1}|$ **do** ▷ Payoff table
    for $j = 1$ to $|P_{r-1}|$ **do**
      Estimate $s_{i,j}$ over k games.
      $P^r_{i,j} \leftarrow s_{i,j}$
    end for
  end for
  $(x_1^r, \ldots, x^r_{|P_{r-1}|}) \leftarrow \text{FP}(P^r)$ ▷ Nash Equilibrium
  $b^r \leftarrow \text{MixtureBot}(x_1^r, \ldots, x^r_{|P_{r-1}|})$
  for $t = 1$ to T **do** ▷ LLM Best Response
    $c_t \leftarrow \text{LLM\_Generate}(b^r)$ ▷ Prompt LLM with b^r
    Estimate $win\_rate(c_t, b^r)$ over k games
  end for
  $b_{\text{new}} \leftarrow \arg\max_t \text{win\_rate}(c_t, b^r)$
  $P_r \leftarrow P_{r-1} \cup \{b_{\text{new}}\}$
end for
Return P_m

3 Results and Conclusions

We evaluate the ability of our procedure to identify increasingly strong strategies in the game. We run the LLM-PSRO procedure, using CodeLlama-7b-Instruct-hf as our LLM, for $m = 5$ rounds in $n = 20$ independent runs, each starting with an initial population of 3 bots (obtained by querying an LLM to produce code for playing in the game in the absence of additional instructions regarding an opponent). We examine the total probability mass placed by the Nash equilibrium on these initial bots, in each round (averaged over the $n = 20$ runs). We expect that as the new population bots grow in strength, less mass would be placed on the original bots, as they lose more and more to the newer generations.

The result of the analysis are shown in Figure 2. The blue line is the total probability mass on the original 3 bots (with the shaded area shown the standard deviation over the differnet LLM-PSRO runs). Even if the quality if the new bots is similar to the original bots, we would expect the mass on the original bots to diminish as mass would also be placed on the new bots; however, if their quality was the same as the bots, we'd expect their total mass to be proportional to their share of the population, which is plotted as a dashed red line in the figure ("Expected mass"). Figure 2 shows that the mass on the original bots diminishes more quickly than their share

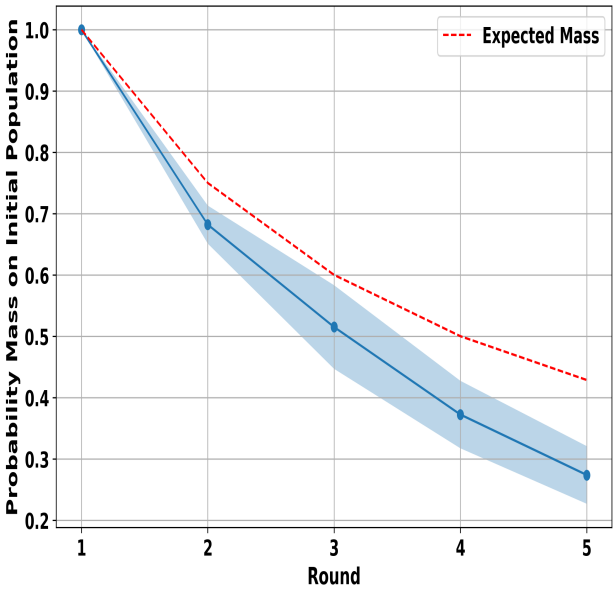

Figure 2: Nash probability mass on initial population of bots, over the LLM-PSRO rounds (averaged over $n = 20$ runs). Lower number reflect more mass on the new generation of bots, indicating that the newer bots are stronger.

of the population, indicating the newly generated bots are of higher quality. In fact, after only 5 rounds, the original 3 bots obtain a mass of only about $\frac{1}{4}$, so the newer bots are mostly optimized against the previous generations of LLM-generated bots. This indicates that the LLM-PSRO method is generating increasingly stronger bots, and very quickly surpasses the quality of the initial population of bots. Our tool lets users select bots for the initial population (using pre-coded strategies, or by typing in strategies). It then runs the process, and allows inspecting the code for the bots added at each iteration. It displays the Nash equilibrium weights in each round, indicating the relative strengths of the bots in each generation.

4 Related Work

Game playing has been a foundational topic in AI since its early days, with classical algorithms such as Alpha-Beta pruning [Knuth and Moore, 1975] and MCTS [Coulom, 2006]. Combining these with deep reinforcement learning has allowed achieving super-human abilities in many games [Campbell *et al.*, 2002; Silver *et al.*, 2018; Perolat *et al.*, 2022] However, these require extensive training and result in game playing policies which are not interpertable by humans [Hernandez-Leal *et al.*, 2019; Angelov *et al.*, 2021]. LLMs offer an alternative approach for game-playing. Researchers used LLMs to model rational players in game theoretic and multi-agent settings [Zhang *et al.*, 2024b; Fan *et al.*, 2024; Guo *et al.*, 2024; Huang *et al.*, 2024]. Some work also uses game theoretic solutions to augment LLMs in strategic settings [Gemp *et al.*, 2024].

In contrast to these approaches we do not ask the LLM to directly play in a game; Such a direct prompting approach can result in suboptimal or inconsistent strategies, highlighting the need for a more structured solution [Guo *et al.*, 2024; Huang *et al.*, 2024]. Rather, in our approach the role of the LLM is to generate a bot (an agent) for playing in the game. As we use the LLM to generate *game playing code*, our method is more akin to LLM software engineering agents (SWE-Agents), that produce piece of software for task [Liu *et al.*, 2024b; Xia *et al.*, 2024; Yang *et al.*, 2024; Nathani *et al.*, 2025]. Some such existing work also combines SWE-Agents with search techniques [Antoniades *et al.*, 2024]. Although we use LLMs to write software, we differ from SWE-Agents in that we allow for competition between the codes based on a game theoretic recipe.

Researchers have employed many game theoretic methods for improving decision making. Classical recipes from game theory such as fictitious play [Hofbauer and Sandholm, 2002] are designed to select minimally exploitable mixed strategies. Recipes based on approximating equilibria have been used for many settings such as security games [Tambe, 2011] and generative models [Aung *et al.*, 2022]. Some such game theory models have been used to optimize the performance of multi-agent reinforcement learning systems such the Stratego board game [Perolat *et al.*, 2022] or the Starcraft video game Nash league [Vinyals *et al.*, 2019]. We leverage the Policy Space Response Oracle framework [Lanctot *et al.*, 2017; Zhang *et al.*, 2024a] but adapt it for code generating LLMs, which is very different from the setting of reinforcement learning for which is has been previously applied.

5 Conclusions

We proposed an approach combining LLM code generation and self-play, that allows for generating bots to play in games. We follow the Policy Space Response Oracle [Lanctot *et al.*, 2017] pattern, but operates in the space of programs where strategies are represented as computer code. By combining LLM bot generation with PSRO, we generate strong opponents whose code is fully readable and interpertable by humans. Hence, the produced bots can be edited and refined by humans, and their decision-making process is transparent.

There are several interesting directions for future work. First, are there alternative algorithms that could achieve a higher performance? Second, we leverage both the code generation ability of LLMs and their strategic reasoning. One might consider ways to strengthen strategic reasoning in LLMs. One option is leveraging synthetic data, such as the outcomes of previous tournaments. Further, we have observed that in many cases the code generation abilities of the LLM could also be improved. Synthetic data produced through the tournaments between the LLM generated programs can also be leveraged to improve the coding ability of the underlying LLMs. Finally, our demo focuses on the game of Checkers, where the agent interaction is competitive (zero-sum) and the environment is deterministic, and the entire state is observed by all the players. How should it be changed for cooperative domains [Nagarajan and Sošić, 2008; Bachrach *et al.*, 2020; Bachrach and Porat, 2010; Sidji *et al.*, 2024] or those with partial observability or increased stochasticity [Bowling and Veloso, 2000; Bachrach *et al.*, 2012; Alsadat and Xu, 2024]?

References

[Allis, 1994] Louis Victor Allis. Searching for solutions in games and artificial intelligence. 1994.

[Alsadat and Xu, 2024] Shayan Meshkat Alsadat and Zhe Xu. Multi-agent reinforcement learning in non-cooperative stochastic games using large language models. *IEEE Control Systems Letters*, 2024.

[Angelov *et al.*, 2021] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(5):e1424, 2021.

[Antoniades *et al.*, 2024] Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. *arXiv preprint arXiv:2410.20285*, 2024.

[Aung *et al.*, 2022] Aye Phyu Phyu Aung, Xinrun Wang, Runsheng Yu, Bo An, Senthilnath Jayavelu, and Xiaoli Li. Do-gan: A double oracle framework for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11275–11284, 2022.

[Bachrach and Porat, 2010] Yoram Bachrach and Ely Porat. Path disruption games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1123–1130, 2010.

[Bachrach *et al.*, 2012] Yoram Bachrach, Reshef Meir, Michal Feldman, and Moshe Tennenholtz. Solving cooperative reliability games. *arXiv preprint arXiv:1202.3700*, 2012.

[Bachrach *et al.*, 2020] Yoram Bachrach, Richard Everett, Edward Hughes, Angeliki Lazaridou, Joel Z Leibo, Marc Lanctot, Michael Johanson, Wojciech M Czarnecki, and Thore Graepel. Negotiating team formation using deep reinforcement learning. *Artificial Intelligence*, 288:103356, 2020.

[Bowling and Veloso, 2000] Michael Bowling and Manuela Veloso. *An analysis of stochastic game theory for multiagent reinforcement learning*. Citeseer, 2000.

[Brown, 1951] George W Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1):374, 1951.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Campbell *et al.*, 2002] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[Fan *et al.*, 2023] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53. IEEE, 2023.

[Fan *et al.*, 2024] Caoyun Fan, Jindou Chen, Yaohui Jin, and Hao He. Can large language models serve as rational players in game theory? a systematic analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17960–17967, 2024.

[Gemp *et al.*, 2024] Ian Gemp, Yoram Bachrach, Marc Lanctot, Roma Patel, Vibhavari Dasagi, Luke Marris, Georgios Piliouras, Siqi Liu, and Karl Tuyls. States as strings as strategies: Steering language models with game-theoretic solvers. *arXiv preprint arXiv:2402.01704*, 2024.

[Griffith, 1974] Arnold K Griffith. A comparison and evaluation of three machine learning procedures as applied to the game of checkers. *Artificial Intelligence*, 5(2):137–148, 1974.

[Guo *et al.*, 2024] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

[Hernandez-Leal *et al.*, 2019] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[Hofbauer and Sandholm, 2002] Josef Hofbauer and William H Sandholm. On the global convergence of stochastic fictitious play. *Econometrica*, 70(6):2265–2294, 2002.

[Hou *et al.*, 2024] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79, 2024.

[Huang *et al.*, 2024] Jen-tse Huang, Eric John Li, Man Ho Lam, Tian Liang, Wenxuan Wang, Youliang Yuan, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Michael R Lyu. How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments. *arXiv preprint arXiv:2403.11807*, 2024.

[Knauer *et al.*, 2020] Henning Knauer, Andrea Dederichs-Koch, and Daniel Schilberg. Developing a reinforcement learning agent for the game of checkers. In *Advances in Human Factors in Training, Education, and Learning Sciences: Proceedings of the AHFE 2020 Virtual Conference on Human Factors in Training, Education, and*

Learning Sciences, July 16-20, 2020, USA, pages 164–169. Springer, 2020.

[Knuth and Moore, 1975] Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1975.

[Lanctot *et al.*, 2017] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[Li *et al.*, 2022] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[Liu *et al.*, 2024a] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36, 2024.

[Liu *et al.*, 2024b] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. *arXiv preprint arXiv:2409.02977*, 2024.

[Mańdziuk *et al.*, 2007] Jacek Mańdziuk, Magdalena Kusiak, and Karol Waledzik. Evolutionary-based heuristic generators for checkers and give-away checkers. *Expert Systems*, 24(4):189–211, 2007.

[Nagarajan and Sošić, 2008] Mahesh Nagarajan and Greys Sošić. Game-theoretic analysis of cooperation among supply chain agents: Review and extensions. *European journal of operational research*, 187(3):719–745, 2008.

[Nathani *et al.*, 2025] Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, et al. Mlgym: A new framework and benchmark for advancing ai research agents. *arXiv preprint arXiv:2502.14499*, 2025.

[Perolat *et al.*, 2022] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

[Schaeffer *et al.*, 2007] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.

[Sidji *et al.*, 2024] Matthew Sidji, Wally Smith, and Melissa J Rogerson. Human-ai collaboration in cooperative games: A study of playing codenames with an llm assistant. *Proceedings of the ACM on Human-Computer Interaction*, 8(CHI PLAY):1–25, 2024.

[Silver *et al.*, 2018] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[Tambe, 2011] Milind Tambe. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press, 2011.

[Vinyals *et al.*, 2019] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

[Xia *et al.*, 2024] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

[Yang *et al.*, 2024] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

[Zhang *et al.*, 2021] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[Zhang *et al.*, 2024a] Ruize Zhang, Zelai Xu, Chengdong Ma, Chao Yu, Wei-Wei Tu, Shiyu Huang, Deheng Ye, Wenbo Ding, Yaodong Yang, and Yu Wang. A survey on self-play methods in reinforcement learning. *arXiv preprint arXiv:2408.01072*, 2024.

[Zhang *et al.*, 2024b] Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Adrian de Wynter, Yan Xia, Wenshan Wu, Ting Song, Man Lan, and Furu Wei. Llm as a mastermind: A survey of strategic reasoning with large language models. *arXiv preprint arXiv:2404.01230*, 2024.