Designing AI Agents with Different Investment Strategies for Board Game Manila — a Reinforcement Learning Approach

Yutong Ren
rentony@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Xinyi Lu
lwlxy@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Jeremy Huang
zjhuang@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

ACM Reference Format:
Yutong Ren, Xinyi Lu, and Jeremy Huang. 2026. Designing AI Agents with Different Investment Strategies for Board Game Manila — a Reinforcement Learning Approach. In . ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In this project, we explore AI agents with different investment strategies for the board game "Manila". We introduce the background/problem description and game mechanics, task overview, and system input and output. We also present the challenges of this task, which we addressed throughout the AI agent design process.

1.1 Problem Description

1.1.1 Background and Game Mechanics: Manila is a popular board game where players take on the roles of smugglers competing to transport goods across the ocean [1]. Simulating the trade in the capital of the Philippines in the 19th century, this game combines elements of gambling and strategic investments. The goal of the Manila game is to accumulate the most wealth among 3 players by the end of the game. Players must balance risk and rewards as they attempt to profit from transporting goods such as silk, ginseng, and jade via the "ships" sailing towards the port of Manila. The game will contain 3 rounds with 3 players, each of whom will possess 30 PESOS (currency) and 3 workers. Initially, there will be 3 ships set to sail towards Manila. Each ship will have 3 spots for placing workers. These 3 ships will progress along a 10-position track based on dice rolls toward the Port of Manila. Players can place one worker per round in the following locations: any of the three ships, ports, or shipyards. In total, there will be 15 distinct spots for worker placement (9 spots on the ship, 3 spots on the ports, and 3 spots on the shipyards). As the ships progress, a roll of the dice determines their movements. In each round, players will deploy their workers and roll dice to move the ships. If a ship reaches the 10th position, it successfully reaches Manila, which then earns profits for the investors in that ship and the corresponding port. If a ship does not reach the 10th position, it will be considered "damaged" and sent to the shipyard for repairs, profiting only the shipyard investor. The entire game follows a strict first-in order: the first port receives the first arriving ship, and similarly, the first shipyard takes in the first failing ship. After three rounds, players tally their money, and the wealthiest player wins.

1.1.2 Challenges: Unlike deterministic board games like Chess or Go, the stochasticity in Manila arises not only from opponent strategies but also from the randomness of dice rolls, the complex state space, and balancing exploration vs. exploitation, making it a rich testbed for studying adaptive AI strategies.

1.2 Our Task Scope

In this work, we will propose AI agents as players for the Manila game, with the goal of winning the game through strategic investments. These three agents will have different "greediness". Specifically, short-sighted agents will prioritize immediate rewards; far-sighted agents will prioritize long-term rewards; moderate agents will balance short-term and long-term rewards. We specify details of the agent design in Section 3. We used a reinforcement learning approach with Q-Learning [17] and Deep-Q Network [2] to implement these agents.

Input and Output: The AI Agents will receive the current game state, incliding the position of the ship, available worker placement spots, players' PESOS, previous investments, and dice roll probabilities. Bounded by the rules, the AI agents will generate decisions on worker placements, investment choices for each round to maximize its wealth. Examples include placing a worker in a ship, investing in a shipyard, or betting on a port.

Contributions: We designed three AI agents with varying levels of "greediness" to explore different investment strategies using reinforcement learning techniques. Specifically, thee agents leverage Q-learning and Deep-Q Networks to learn optimal strategies. We also analyzed the performance of these agents to assess how varying greedinesses impact their success. We believe that this exploration advances the understanding of AI strategies in stochastic, multi-agent environments, and adaptive decision-making. All team members contributed equally on environment simulation, agent design, agent training, testing, and comparative analysis.

2 Related Work

2.1 Q-Learning and Its Application in Game Agents

Q-learning, a model-free reinforcement learning algorithm, has been extensively applied to various board games, enabling agents to learn optimal strategies through iterative updates of Q-values. This approach has been successfully implemented in games such as Tic-Tac-Toe, Mancala, [7], and Dominion (a card game [3]), where agents are evaluated based on their win rates against opponents employing

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

random actions. The flexibility of Q-learning makes it suitable for diverse game simulations, as it does not impose constraints based on task type.

However, maintaining a comprehensive Q-table—which stores expected rewards for each state-action pair—can become computationally expensive, especially in games with large state spaces. In this work, we plan to apply Q-learning to develop AI agents capable of strategic investment and optimized decision-making in the stochastic environment of Manila. To address the computational challenges, we will design efficient state representations that capture key game elements, such as ship positions, investment slots, and financial standings. Through self-play, the agents will iteratively refine their strategies, adapting to the probabilistic mechanics of the game. This approach aims to evaluate how well Q-learning can model strategic decision-making and simulate human-like gameplay in Manila.

2.2 Deep-Q Networks and Its Application in Board Games

To deal with the exponentially increasing environment state size, Deep-Q Networks (DQN) combines the deep neural network with the Q-learning approach to understand high-dimentional sensory inputs and manage complex decision making [4, 6, 11]. Starting with Atari [10], DQN-based agents has reaching higher performance than other AI agents and human agents games from various genres, including board games [8, 9, 12], strategic games like StarCraft 2 and Overcooked [14, 16], and even 3D games like Minecraft [15].

Having high-deminsional environments where diverse and complex strategies are involved, one line of work has focused on developing DQN-based agent for board games [5, 9, 11]. Given the well-defined reward functions and limited action states, DQN has shown its advanced capacity in learning optimal strategies in various board games including Texas Hold'em and Mahjong [5, 8]. Integrated with search algorithms like Monte Catlo Tree Search, AlphaZero started off with random play and learned to achieve superhuman performance [12]. With a variation of DQN, AlphaGo outperformed top human players, further illustrate the learning ability of DQN and its adaptability to unseen scenarios [13]. However, due to the low explainability nature of deep neural network, it remains hard to explain the learned strategy in DQN-based agents. In this work, if time permits, we will implement DQN to address the complexities of the game environment, which may be better handled than with Q-learning, potentially enabling faster convergence in agent training.

3 Deep Dive Into the Environment

The game environment of Manila can be encoded into a multi-agent stochastic and discrete state space S, which consists of:

- The available investment slots and associated costs in the three ports, shipyards, and ships.
- The positions of the three ships on the board.
- The difference of money between players.

Thus, the states can be represented with vectors with 17 dimensions, including 1 dimension for each ship, port or shipyard recording the remaining spots, 3 dimensions in total for the position of the ship, and 3 dimensions for the difference of money between 2 players.

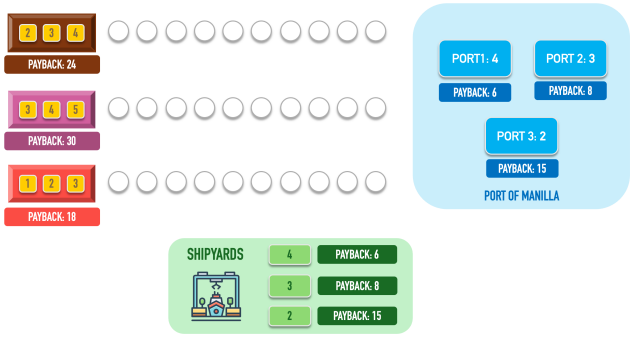

Figure 1: Visualization of the Simulated Manila Game Environment.

To take into account the progress of the game, we also included 1 dimension for the round number and 1 dimension for the latest action. Each ship is represented as an instance of a `ship` class with attributes such as: `name`, `cost`, `payback`, `position`, `available`, etc. To represent the three worker spots on an individual ship, we used a three-element array containing the costs for each spot (e.g., `[3,4,5]`). The agents do not get to choose which spot they want to place their workers in each individual ship (`ship` instance). For example, the agent who places a worker on an empty Ship 2 will only be able to place the worker in the first spot; the next agent who places a worker on the Ship 2 can only fill in the second spot.

The shipyard, represented as the `shipyard` class, contains attributes such as `name`, `cost`, `payback`, `available`, and `investors`. The port (represented by the `port` class), follows a similar structure as shipyard.

In terms of computing the currency, instead of recording the absolute value of money, we recorded the difference between their money and other agents' because this is a competitive game. Using relative values not only tells us more about the game states, but is also computationally efficient: the range relative values is much narrower than absolute values, thus needing less training epochs.

Our Manila game follows the following configurations:

Ships(`shipName`, `costs_arr`, `payback`):

- First ship: (`ship1`, `[2,3,4]`, 24)
- Second ship: (`ship2`, `[3,4,5]`, 30)
- Third ship: (`ship3`, `[1,2,3]`, 18)

Ports(`portName`, `cost`, `payback`):

- First port: (`port1`, 4, 6)
- Second port: (`port2`, 3, 8)
- Third port: (`port3`, 2, 15)

Shipyards(`shipName`, `cost`, `payback`):

- First shipyard: (`shipyard1`, 4, 6)
- Second shipyard: (`shipyard2`, 3, 8)
- Third shipyard: (`shipyard3`, 2, 15)

The agents' action space A includes placing a worker at one of the available positions or skipping the round. If the agent decides to place a worker, the candidate positions contain Port 1, Port 2, Port 3, Shipyard 1, Shipyard 2, Shipyard 3, Ship 1, Ship 2, and Ship 3 if they have space left. Particularly, each agent will take turns to

take action in the environment. They will observe the actions of other agents and the progress of each ship determined by a roll of dice.

4 Methodology

4.1 Q-Learning and Deep-Q Learning

For this work, we train agents with Q-learning, which is model-free learning, to reduce the number of training data needed. We used the following formula to update the Q-values in the Q-tables:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(R(s) + \gamma \max_a Q(s',a')\right)$$

We tuned several hyper-parameters to optimize the training process and ensure favorable learning outcomes. The learning rate, denoted as α, was set to 0.2 to facilitate gradual updates to the Q-values, enabling a balanced approach between exploration and exploitation. The discount factor, γ, was set to 0.9 in the Q-function, which allows the agent to appropriately weigh future rewards against immediate rewards in its decision-making process.

In addition to Q-learning, we also employed DQN to address challenges posed by Manila's large state space. Instead of maintaining a Q-table, DQN uses a neural network to predict Q-values $Q(s,a;\theta$, where θ represents the parameters of the network. The parameters were updated to minimize the difference between predicted Q-values and target Q-values (loss function):

$$L(\theta) = \mathbb{E}(s,a,r,s')\left[\left(R(s) + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta)\right)^2\right]$$

4.2 Reward Design

For this game setting, we set the immediate rewards to be the immediate cost of agents' investments. For example, when an agent places a worker in an empty Ship 1, the immediate reward will be -2, since it needs 2 PESOS to invest in the first slot on ship 1. Similarly, when an agent take the action of port 1, the reward is -4 since it takes 4 PESOS to invest in port 1. The long-term rewards, captured by the term $\gamma \cdot \max_a Q^*(s_{\text{next}})$, will be the potential future payoff if the ship successfully reaches the harbor.

4.3 Agent Training

We employed self-play to train the game AI agent, where three agents, each utilizing a strategy (from far-sighted, short-sighted, moderate), competed against one another over the course of three rounds. Each round contains an investment phase followed by ship movement that was determined by a dice roll. Upon completion of the three rounds, the game concluded, and the final monetary outcomes were calculated, representing the end of an episode.

Actions, states, and rewards are recorded to form state-action-reward sequences, which are then used to update the agent's policy and value networks. The policy network optimizes action selection for favorable outcomes, while the value network improves state evaluation. This cycle of self-play, data collection, and network updates continues until the agent achieves consistent performance improvements.

To ensure robust learning, exploration techniques are applied to prevent premature convergence on suboptimal strategies. Self-play enables the agent to autonomously learn and refine strategies, achieving a high level of competence through iterative, self-driven adaptation.

4.4 Action-Selection Policy

To ensure robust learning, we included exploration techniques to prevent premature convergence on suboptimal strategies. We utilized the ϵ-greedy strategy, wherein the agent balances exploration and exploitation by choosing the best-known action with a probability of ϵ and exploring other actions with probability $1-\epsilon$. This allows the agent to systematically explore its environment while also exploiting the most promising actions based on its current knowledge.

For this work, we developed agents with three different behaviors: short-sighted, moderate, and far-sighted. The types of behaviors were determined by an additional term we introduced in the agents' decision making process. In traditional Q-learning models, the agent should choose the optimal action based on the value in the table, choosing the action with the maximum Q-value. To simulate the behavior of short-sighted, moderate, and far-sighted players, where their perceived rewards of actions are not exactly the same as the real rewards, but also related to the risk they would have to take, we introduced an additional term proportional to the expectation reward of action. The expectation reward is calculated based on the possibility that the player can get the reward times the amount of reward they could get out of it. We represent their perceived rewards $Q^*(s,a)$ which influence agents' decisions using:

$$Q^*(s,a) = Q(s,a) + fE[r(x,a)]$$

We developed different agents with different values for f, in the equation. As f increases, the agents' decision-making will become more far-sighted, as they will prioritize the long-term rewards and are more likely to take actions that may not give an immediate payback but are expected to yield higher cumulative rewards over time. Conversely, when the value of f is low, the agents will be more short-sighted and focus on avoiding immediate costs only. In this round, we selected $f = 0$ for the short-sighted agent, which represents that it doesn't perceive any extra value in future rewards at all. It is expected to show a tendency to avoid all risky behaviors. We selected $f = 1$ for the moderate agent, representing that their perceived value of future rewards corresponds to the reward expectation of the action. And the far-sighted agent has $f = 10$ to show that they are more willing to take actions leading to higher future rewards regardless of the risks. Since $f = 0$ represents the same decision-making method as in original Q-learning method, we are planning to use the short-sighted agent ($f = 0$) as baseline. The training process with DQN followed the same principles as Q-learning.

5 Experiment and Evaluation

We adopted both Q-learning and DQN to train the agents. For both methods, we experimented with different numbers of epochs, starting from 1000, and gradually increasing. First, using Q-learning, we trained each agent separately for 50,000 episodes. Since each game contains 3 rounds, and the reward are provided after all 3 rounds, the training was iterated for 150,000 steps. We adopted $\alpha = 0.2$ to balance between exploration and exploitation, and $\gamma =$

Table 1: The moderate agent got significantly higher winning rates than the short-sighted agents during the training.

Short-Sighted Agent 1	Moderate Agent	Short-Sighted Agent 2
6.43%	89.01%	4.56%

Table 2: The far-sighted agents got significantly higher winning rates than the short-sighted agents during the training

Short-Sighted Agent 1	Far-Sighted Agent	Short-Sighted Agent 2
6.26%	89.34%	4.40%

0.9 for weighing future rewards. Then using DQN, we found that training the agents for about 10000 episodes already resulted in convergence.

We evaluated the performance of three distinct AI agents through multiple simulations of Manila, recording the *win rate* across 50000 games. Our key metric will be the percentage of games won by each agent type. We will also observe the convergence behavior of the agents, focusing on the stability and speed of convergence in learning an optimal strategy.

5.1 Q-learning

5.1.1 Training. Through Q-learing, both the moderate agent and the far-sighted agent learned better strategy than the baseline. As shown in Table 1 and Table 2, the moderate agent and the far-sighted agents got significant higher winning rates during the 50000 training episodes.

This conclusion is also supported by the convergence plots and the winning rate. As shown in Figure **??**, the delta Q value of the moderate agent constantly decrease as the number of steps increases. Figure 3 shows the convergence plot of the far-sighted agent in a similar trend. This shows evidence of the effectiveness of our modeling in different agents' behaviors and also in learning the optimal strategy. As shown in Figure 4, the moderate agent is learning to win the short-sighted agents as the training proceeds. The winning rate increased rapidly and reached 85% until 10000 episodes. Then the increase slowed down. That means, although more episodes is preferrable, 10000 episodes would lead to an agent that has adequate knowledge and strategies for the game.

5.1.2 Performance of Different Investment Strategies in Manila. We let the three agents play against each other 50000 times and calculate their winning rate. A higher winning rate indicates that the agent has learned more optimal strategies for the game.

Running our current implementation of the Manila game, we observed that the agent's performance demonstrated distinct behavioral differences based on the f values (corresponding to short-sighted, moderate, and far-sighted agents), as shown in Table 3. After training for 50000 epochs, moderate and far-sighted agent got significantly higher winning rates than the short-sighted agent ($f = 0$), with the moderate agent wining more. Since the short-sighted agent has the same decision making function as the original Q-learning, this result gives evidence of the effectiveness of our modeling in different agents' behaviors and also in learning the optimal strategy. It also indicates that the willingness to take

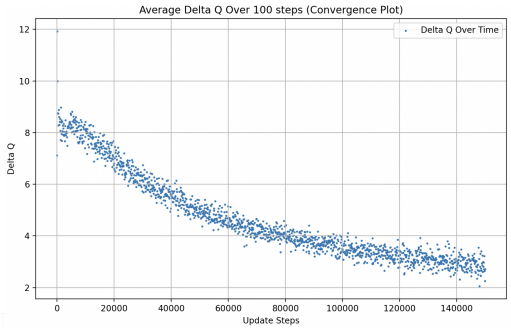

Figure 2: Convergence plot for the moderate agent shows that the delta Q value constantly decreased as the number of steps increases

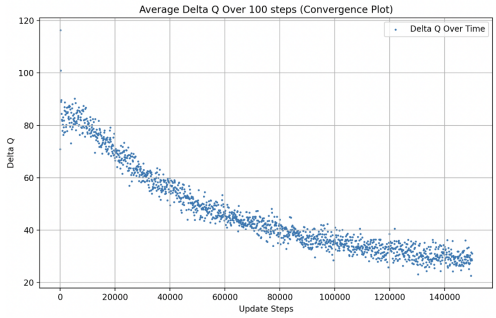

Figure 3: Convergence plot for the far-sighted agent shows that the delta Q value constantly decreased as the number of steps increases

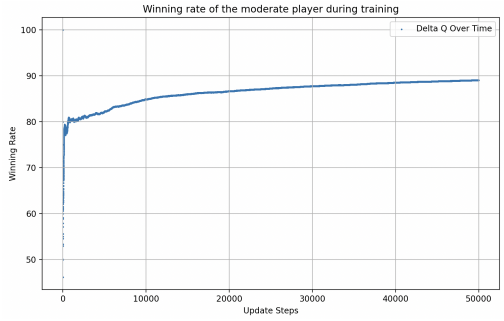

Figure 4: The winning rate of the moderate agent increased and reached 85% after 10000 episodes

risk is an essential strategy for winning in Manila. Although being overly greedy would lead to a lower winning rate than the moderate players, it is preferable to a short-sighted agent.

Moreover, we delved into the dynamics of gameplay when three agents, and we found that the order of play influences the winning rate. Player always get a slightly higher winning rate when they are the first to take action, than when the same agent is in the second or the third order. All the agents showed the same trend.

Table 3: The winning rate of the three agents with different orders after training for 10000 epochs. The moderate and far-sighted agents always get significantly higher winning rates, indicating the effectiveness of our modeling in different agents' behaviors.

Order	Short-Sighted Agent	Moderate Agent	Far-Sighted Agent
123	4.84%	55.97%	39.19%
132	4.27%	51.84%	43.89%
213	3.39%	56.68%	39.93%
231	2.83%	56.48%	40.69%
312	3.25%	46.74%	50.01%
321	2.66%	46.65%	50.69%

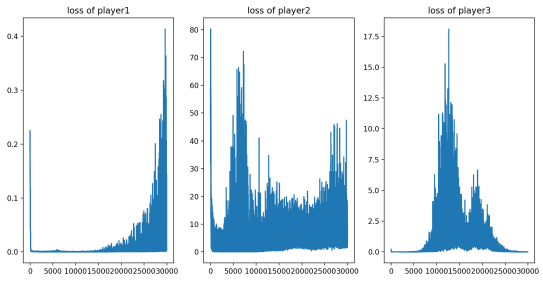

Figure 5: After 10000 episodes, the loss of the far-sighted agent already converged.

This phenomenon is likely due to the fact that the initial player possesses a wider array of investment options,

5.2 DQN

To address the limitations of Q-learning in managing large state spaces, we implement DQN. First, we tried with the self-play training, but we found that for DQN, the naiveness of short-sighted agent (baseline) limited the learning of moderate and far-sighted agents. So we tried to include all three agents in the training phase. As shown in Figure 5, after training for 10000 epochs, the loss of player 3 (far-sighted agent) already converged, while the loss of player 1 (shortsighted agent) keep increasing, indicating that player 1 didn't manage to learn effective strategies throughout the training. The loss of player 2 (moderate agent) did not converge. This is probably because it is playing against the far-sighted agent.

5.2.1 Performance of Different Investment Strategies in Manila. Although the far-sighted agent got the lease loss, we found that the moderate agent got a slightly higher winning rate than the far-sighted agent, as shown in Table 4. Compared with the result from Q-learning, where the far-sighted agent learns significantly better strategies than the short-sighted agent, while the winning rate is much lower than the moderate agent, this result indicates that with a more powerful model to learn the strategy, the far-sighted agent has the potential to learn strategies that are compatible with the moderate agent. This result aligned with our expectation that as DQN replaces the Q-table with a neural network to approximate the Q-function, enabling efficient generalization across complex state-action pairs, it is particularly advantageous in environments like Manila, where the state space is highly dynamic and has the stochastic nature of dice rolls and the actions of competing players further add to the complexity.

Table 4: Both the moderate agent and the far-sighted agent got higher winning rates than the short-sighted agents

Short-Sighted Agent	Moderate Agent	Far-Sighted Agent
6.98%	46.90%	46.12%

References

[1] [n. d.]. Manila. https://boardgamegeek.com/boardgame/15817/manila

[2] [n. d.]. Reinforcement Learning (DQN) Tutorial — PyTorch Tutorials 2.4.0+cu121 documentation. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[3] George Angelopoulos and Dimitris Metafas. 2021. Q Learning applied on the Board Game Dominion. In *Proceedings of the 24th Pan-Hellenic Conference on Informatics (PCI '20)*. Association for Computing Machinery, New York, NY, USA, 34–37. https://doi.org/10.1145/3437120.3437269

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

[5] Johannes Heinrich and David Silver. 2016. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121* (2016).

[6] Chengpeng Hu, Yunlong Zhao, Ziqi Wang, Haocheng Du, and Jialin Liu. 2024. Games for Artificial Intelligence Research: A Review and Perspectives. *IEEE Transactions on Artificial Intelligence* (2024).

[7] Peter Jamieson and Indrima Upadhyay. 2022. A Technique to Create Weaker Abstract Board Game Agents via Reinforcement Learning. https://arxiv.org/abs/2209.00711v1

[8] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. 2020. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590* (2020).

[9] Yudong Lu, Youpeng Zhao, Wengang Zhou, Houqiang Li, et al. 2023. Danzero: Mastering guandan game with reinforcement learning. In *2023 IEEE Conference on Games (CoG)*. IEEE, 1–8.

[10] Volodymyr Mnih. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[12] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

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

[14] Yuhang Song, Jianyi Wang, Thomas Lukasiewicz, Zhenghua Xu, and Mai Xu. 2019. Diversity-driven extensible hierarchical reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4992–4999.

[15] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.

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

[17] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (May 1992), 279–292. https://doi.org/10.1007/BF00992698