Learning Monopoly Gameplay: A Hybrid Model-Free Deep Reinforcement Learning and Imitation Learning Approach

Marina Haliem[1*], Trevor Bonjour[1*], Aala Alsalem[1], Shilpa Thomas[2], Hongyu Li[2], Vaneet Aggarwal[1], Bharat Bhargava[1], and Mayank Kejriwal[2]

***Abstract*—Learning how to adapt and make real-time informed decisions in dynamic and complex environments is a challenging problem. To learn this task, Reinforcement Learning (RL) relies on an agent interacting with an environment and learning through trial and error to maximize the cumulative sum of rewards received by it. In multi-player Monopoly game, players have to make several decisions every turn which involves complex actions, such as making trades. This makes the decision-making harder and thus, introduces a highly complicated task for an RL agent to play and learn its winning strategies. In this paper, we introduce a Hybrid Model-Free Deep RL (DRL) approach that is capable of playing and learning winning strategies of the popular board game, Monopoly. To achieve this, our DRL agent (1) starts its learning process by imitating a rule-based agent (that resembles the human logic) to initialize its policy, (2) learns the successful actions, and improves its policy using DRL. Experimental results demonstrate an intelligent behavior of our proposed agent as it shows high win rates against different types of agent-players.**

***Index Terms*—Monopoly, Deep Reinforcement Learning, Rule-Based Agent, Imitation Learning**

I. Introduction

Despite numerous advances in deep reinforcement learning, the majority of successes have been in two-player, zero-sum games, where it is guaranteed to converge to an optimal policy [1], such as Chess and Go [2]. Rare (and relatively recent) exceptions include Blade & Soul [3], no-press diplomacy [4], Poker[1] [6], and StarCraft [7]. There is already some evidence emerging that pure deep reinforcement learning may not always be the only (or even best) solution for multi-player games with a distinctly game-theoretic component. For example, the system used in [6] for multi-player Poker games, relied on improved Monte Carlo counterfactual regret minimization.

In particular, there has been little work on agent development for the full 4-player game of Monopoly, despite it being one of the most popular strategic board games in the last 85 years. Exceptions include [8] and [9], but even in these, the authors consider an overly simplified version of the game, and neither work considers trades between players.

Monopoly is a turn-based game, where players take turns by rolling two six-faced dice and act according to the square they land on. In Monopoly, players come across as landowners who seek to buy and sell a set of properties. The winner is the one who forces every other player into bankruptcy and thus, achieving a monopoly over the real estate market.

Mainly, due to the complications involved in the nature of the multi-player monopoly game where players have to make several decisions every turn, it imposes several challenges when it comes to representing Monopoly as a Markov Decision Process (MDP). First of all, it imposes a vast state space and a highly stochastic transition function. It also involves randomness, as players roll the dice, occasionally draw cards and act accordingly. In addition, it is an imperfect information game [10] [11] - the order of the chance and community chest cards is unknown, and incomplete information - the opponent strategies are not known to the player. Monopoly also involves complex actions, such as: making trades, which makes the decision-making harder - what to offer up for a trade, whom to offer it to, which offers to accept/reject, etc. Thus, Monopoly introduces a highly complicated task for an RL agent to play and learn its winning strategies [12].

In this paper, we implement strong rule-based agents that reflect successful tournament-level strategies adopted by actual human players, including (i) a preference for purchasing (or acquiring through trading) all four railroads, (ii) a preference for acquiring the *Oriental Avenue*, *Vermont Avenue*, and *Connecticut Avenue* (i.e., ‘Sky Blue’ and most rewarding to own) properties, as well as (iii) a preference for purchasing certain other high-reward property groups (especially, the orange property group).

Besides, in this work, we utilize the RL approach for modeling Monopoly as an MDP, allowing RL agent-

[1] Purdue University, [2] University of Southern California, [*] Equal contribution

[1]We note that, even in this case, a two-player version of Texas Hold ’em was initially assumed [5] but later superseded by a multi-player system.

players to learn to play and win. We present a hybrid deep reinforcement learning agent that is capable of (1) imitating the strong rule-based agent, to initialize its strategy and start the learning process with a strong policy. (2) improving its policy using deep Q-learning that enables it to learn the successful actions and thus, develops a winning strategy against four baseline agent-players, one using a random policy, two with a fixed, rule-based approach, and one using traditional Q-learning.

The key contributions of this paper can be summarized as follows:

- Modelling Monopoly as an MDP problem including most of the universally adopted game-play rules for Monopoly.
- Implementing three rule-based agents that reflect successful human logic strategies for winning Monopoly.
- Developing a hybrid RL agent that initializes its policy by imitating a strong rule-based agent that resembles human logic. This step helps our agent to converge to the optimal policy faster than the case if it starts its learning from scratch.
- Then, our RL agent improves its policy using deep Q-learning to learn a winning strategy of playing Monopoly.
- Our evaluation results show that our RL agent beats the baselines of several settings. It outperforms the smart baselines (that mimics the human logic) by a margin of 15-25% in the percentage of game-wins and about 40% - 60% in the percentage of tournament-wins.

The rest of this paper is organized as follows: Section II discusses the work in literature related to Monopoly and multi-agent games. In Section III, we describe the rules of the Monopoly game enforced in our simulator. Then, Section IV explains our proposed three rule-based agents for playing Monopoly. Section V describes our proposed hybrid approach for learning to play and win Monopoly. In Section VI, we discuss our experimental settings and results. Finally, Section VII concludes our paper.

II. Related Work

Despite the all-time popularity of the game, an RL approach for agents learning to play Monopoly has not been sufficiently studied in literature. To the best of our knowledge, there is very little related work in this regard. In [8], authors proposed a novel representation of the famous board game Monopoly as a Markov Decision Process (MDP). There are some older attempts to model Monopoly as Markov Process including [13]. However, these attempts only considered a very simplified set of actions that players can perform (e.g., buy, sell, do nothing). In [8], an RL agent is trained and tested with two different players, and a Q-learning strategy is employed along with a neural network. In recent work [9], authors apply a feed-forward neural network with the concept of experience replay to learn to play the game. However, their approach supports the idea that there is no one strategy that will always win against any other strategy while maintaining high win-rates.

Starting from its initial state until the end, all possible moves of any game, especially turn-based games (e.g., board games) can be represented in a directed graph. Nodes of the graph represent a specific possible state of the game, and edges represent the transition from one state to another. This structure is called a game tree. Game trees could be used to find the best moves in a game, but they would get large very quickly. Hence the search operations become quite slow which is undesirable. In [12], Monte Carlo Search Tree method is applied to the game tree of Monopoly. The same method is used in the software program AlphaGo to play the board game Go [2]. Monte Carlo methods rely on random sampling, where a game is played out many times from start to end with random moves to discover the best possible moves. This way the game tree could be minimized as much as possible for speed-up and lesser memory usage. However, in Monopoly, the distribution of actions is skewed where certain actions tend to occur more frequently than others, which results in an unbalanced game tree.

In the Monopoly game, a move consists of different action types such as: buying, selling, mortgaging, improving, offering a property, as well as skipping a player's turn, and using cards or money to get out of jail. While the action type "skip turn" has no further details, a "buy" operation consists of many parameters such as: who to buy from, how much to pay, and which property to buy. Because of this, Monte Carlo algorithm encounters actions with multiple parameters more than those with lesser parameters within the whole possible action space. Thus, the idea represented in [12] is to separate the choice of action types (e.g., buy, sell, skip turn) and the actual action parameters (e.g., options of chosen action type) from each other. So, the Monte Carlo Search Tree method collects chosen action types and actions separately from the randomly played games, which results in a more balanced game tree. In [12], authors claim that following this methodology has made the Monopoly playing faster, allowed to remove restrictions applied in the algorithm to get a decent performance in the single-step methodology, and get better game results. In [14], the same methodology is applied for the game Settlers of Catan board game, and similar successful results are indicated.

To mitigate the issue of unbalanced action distribution, our proposed approach does not adopt random sampling for exploration; instead, we initialize our agent's policy using a strong rule-based agent to provide a deterministic option discovery to aid exploration.

Furthermore, unlike previous work [9], [8], we do not limit the action space in Monopoly to buy, sell and do nothing. Instead, we consider all possible actions (Table I), including trades, to make the game as realistic as possible. This makes the task more challenging since we now need to deal with a high-dimensional action space. To handle a larger action space, we propose a novel hybrid approach that combines a rule-based strategy with deep Q-learning. This hybrid agent, is trained with and then tested against four other baseline players. Each opponent uses a different strategy to play its game. Thus, the novelties in this paper include the definition of the state and action spaces of the game, the reward function, the number of players as challengers, and the use of a strategy that stabilize the learning scheme while establishing a strong initial policy using imitation learning. In addition to this, we propose three rule-based agents (explained in Section IV) that reflect winning strategies adopted by real human players. One of these strong rule-based agents will, then, serve as the initial strategy for our RL agent. Using imitation learning, our DRL agent establishes a strong start policy instead of learning from scratch (explained in Section V).

III. Monopoly Game

Monopoly is a turn-based board-game where four players take turns by rolling a pair of unbiased dice and make decisions based on the square they land on. Figure 1 shows the conventional Monopoly game board that consists of 40 square locations. These include 28 property locations, distributed among 8 color groups (22 "Real Estate" properties), 4 railroads, and 2 utilities, that players can buy, sell, and trade. Additionally, there are two tax locations that charge players a tax upon landing on them, six card locations that require players to pick a card from either the *community chest* card deck or the *chance* card deck [12], the *jail* location, the *go to jail* location, the *go* location, and the *free parking* location. Our game schema also specifies all assets, their corresponding purchase prices, rents, and color. The purchase prices are shown in every square that corresponds to an asset in Fig. 1.

The rules of the game are very similar to the conventional Monopoly game rules [2]. A brief of these rules is included in the Appendix. At startup, each player gets $1500 in cash, and all remaining cash and other equipment go to the Bank. On a player's turn, the player must roll the dice and move his/her token forward the number of spaces as rolled on the dice. Players can do a number of trades (e.g. building improvements, etc.) at the start of their turn before rolling the dice. Some of

[2]https://www.hasbro.com/common/instruct/monins.pdf

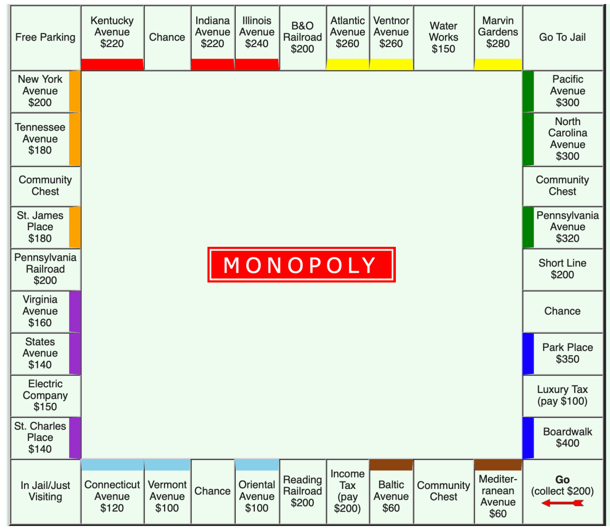

Fig. 1: Monopoly Game Board

the rules lead to increased complexity and stochasticity while others lead to race conditions. To control for these, the following modifications have been made to the rules of the games:

- **Game Phases**: The game is divided into three phases: *pre-roll, post-roll, and out-of-turn*. These are elaborated in more detail in section III-A.
- **Doubles**: The rules associated with rolling doubles as in the US version of the board game have not been considered. Doubles are treated similar to any other dice roll since they do not have any effect on any of the game aspects.
- **Get out of Jail**: If a player ends up in jail, it may use the *Get out of Jail Free* card (if it has one), pay a jail fine of $50 to get out of jail, or may decide to *skip the turn* and remain in jail for that round of the game. As previously mentioned, rolling doubles are not treated differently, and cannot be used to get out of jail as in a default US version of the board game.

A. Game Phases

In an ordinary game of Monopoly, all active players (i.e., players that have not lost) are allowed to take certain actions like mortgaging their property or improving their property even when it is not their turn to roll dice. If multiple players take actions simultaneously, the game can become unstable. To avoid this and to be able to keep track of all the dynamic changes involved in the game, the game has been divided into three phases: *pre-roll, post-roll, and out-of-turn*.

The player whose turn it is to roll the dice might want to take some actions before the dice roll. These actions are taken in the *pre-roll* phase. Once the *pre-roll* phase has concluded for the player whose turn it is to roll the dice, the other players are given the opportunity to take actions before this player rolls the dice. This phase is called the *out of turn* phase. Every player is allowed to take actions in a round-robin manner in this phase until all players decide to *skip the turn*, i.e., does not want to take actions in that phase, or a pre-defined number of *out of turn* rounds have been completed. The player whose turn it is to roll the dice has to now roll the dice. The player's position is updated to the sum of the number on the dice and this player enters the *post-roll* phase where it can take actions based on its new position after the dice roll. This phase is exclusive to the player who rolled the dice.

Below, we specify the action choice associated with each game phase:

- **Pre-roll Phase:** mortgage property, improve property, use get out of jail card, pay jail fine, skip turn, free mortgage, sell property, sell house or hotel, accept sell property offer, roll die, make trade offer, accept trade offer.
- **Post-roll Phase:** mortgage property, buy property, sell property, sell house or hotel.
- **Out-of-turn Phase:** free mortgage, sell property, sell house or hotel, accept sell property offer, make trade offer, accept trade offer, skip turn, mortgage property, improve property.

If a player has a negative cash balance at the end of their post-roll phase, they get a chance to amend it. If they are unsuccessful in restoring their cash balance to 0 or positive, bankruptcy procedure will begin and the player loses the game.

B. Trading

Trading is a very important action that players can use to exchange properties and/or cash to one or more players. The following are the rules of trading:

- Players can trade only un-improved and un-mortgaged properties.
- Players can make trade offers simultaneously to multiple players. The player to whom the trade offer is made is free to accept or reject the offer. The trade transaction gets processed only if the player accepts the trade offer. Once a trade transaction is processed, all other simultaneous trade offers for the same property are terminated.
- Any player can have only one outstanding trade offer at a time, and no other player can make an offer till the pending offer is accepted/rejected. In other words, an existing pending offer has to be accepted or rejected before another trade offer can be made to this player.

IV. Proposed Rule-Based Approach for Monopoly

We propose rule-based agents that, in addition to buying or selling properties, are also capable of making trades. The rules are based on successful tournament-level strategies adopted by actual human players, including (i) a preference for purchasing (or acquiring through trading) all four railroads, (ii) a preference for acquiring the *Oriental Avenue*, *Vermont Avenue*, and *Connecticut Avenue* (i.e., 'Sky Blue' and most expensive) properties, and (iii) a preference for purchasing certain other high-reward property groups (especially, the orange property group). Several informal sources on the Web have documented these strategies though they do not always agree[3]. A full academic study on which strategies yield the highest probabilities of winning has been lacking, perhaps because the complex rules of the game have made it difficult to analytically formalize.

In the following sections, we present the details of two strong rule-based agents along with the details of another simple rule-based agent that we use as a baseline in our experiments.

A. Simple Baseline Agent

This is a very simple agent that does not think too much before making a decision. During the *pre-roll phase*, the agent ideally skips its turn. In the event that its associated player is in jail, then the agent tries to free that player from jail by either using the *Get out of jail free* card if it has one or by paying the jail fine. During the *post-roll phase*, the agent decides if its player should buy an un-owned property upon landing on it or not. The agent strategizes the most during the *out of turn phase*. In this phase, the agent makes and accepts *one way trade offers*, i.e., those that involve only one property in exchange for cash. It accepts an open trade offer that involves buying a property from another player in return for cash if accepting the offer results in a monopoly. It also tries to make a trade offer to another player by offering a property in return for cash when its cash balance is low. However, it is not capable of making trade offers that involve exchange of properties. The agent also makes the decision to free a mortgage or improve a property by building houses and hotels if the player has enough cash balance required to do so.

The agent is also capable of bidding when the bank puts up a property for auction. The agent has to bid an amount greater than the current bid in order to stay in the auction. If the current bid b is lower than the price of the property p_i, then the new bid amount $\hat{b}$ is set to: $b + \frac{p_i - b}{2}$. If he current bid b satisfies $b \geq p_i$, the

[3]Two resources include http://www.amnesta.net/monopoly/ and https://www.vice.com/en/article/mgbzaq/10-essential-tips-from-a-monopoly-world-champion.

agent bids at a price greater than b, only if by obtaining that particular property, the agent gets all the properties in that color group and thus acquire a monopoly. If the player has a negative cash balance, the agent attempts to handle it by taking actions in the following order: selling improvements back to the bank, mortgaging properties, or selling properties back to the bank. If none of these actions result in a non-negative cash balance, the player goes bankrupt and loses the game.

B. Sophisticated Rule-based Agent

By designing this agent, we adopt a human-in-the-loop concept where we utilize the human-player logic towards winning this game to be our RL agent's initial strategy.

This agent has been built on top of the Simple Baseline Agent with more sophisticated heuristics and strategies. These are rules tried and tested by players when playing monopoly tournaments. Trading has been found to be a very effective strategy in improving player performance if strategized well. Besides *one way trade offers*, this agent is capable of making *two way trade offers* that involve the exchange of properties with/without the involvement of cash between players. It is also capable of rolling out trade offers simultaneously to multiple players. By doing so, the agent increases the probability of a successful trade, so it can acquire properties that lead to monopolies of a specific color group more easily. To yield a higher cash balance, the agent aggressively seeks to improve its monopolized properties (by building houses and hotels). Making thoughtful simultaneous trade offers that involve offering properties of low value to the player who is making the trade offer but of high value to the player to whom the offer is being made and vice versa while requesting properties in the trade offer has a high offer acceptance rate. This has shown to significantly improve performance compared to v1-agent. In the event that its associated player ends up with low or negative cash balance, this agent adopts better heuristics to save its player from bankruptcy. This agent makes stronger checks before mortgaging and selling properties and/or improvements to ensure that the sale helps it improve its cash balance.

C. Smarter Rule-based Agent

This agent adopts all the logic described in the previous sophisticated agent (Section IV-B). On top of that, this agent would aggressively buy/bid/trade for all four railroads and any properties in the Orange color set (*St. James Place*, *Tennessee Avenue*, *New York Avenue*) or in the Sky Blue color set (*Oriental Avenue*, *Vermont Avenue*, *Connecticut Avenue*) that are the most rewarding on the board. Once this agent lands on one of these locations and no other player owns it, it would try to buy it, even if the agent has to sell some other properties that it currently owns. If the bank auctions any of these locations, this agent would attempt to buy it for the lowest price possible by making minimal increments to the current bid. This agent also prioritizes buying properties that would lead to monopolizing any color group, if/when it lands on them. Furthermore, this agent persistently makes a *trade offer* to other players that currently own one of the four *railroads*, or any of the properties in the Orange or Sky Blue color set. Also, this agent targets to monopolize two color sets at the same time (Orange and Sky Blue) and will prioritize acquiring properties that belong to these color groups (Orange group is given a higher priority).

Algorithm 1 Deep Q-learning with experience replay

Initialize replay buffer D, policy Q-network parameters θ and target Q-network parameters $\hat{\theta}$.
for $e = 1 : Episodes$ **do**
  Initialize the game board with arbitrary order for player turns.
  Get initial state s_0
  for $t = 1 : T$ **do**
    With probability. ϵ, **select** action a_t by imitating rule-based agent
    Else $a_t \leftarrow \text{argmax}_a Q(s_t, a; \theta)$
    Execute action based on a_t
    Calculate reward r_t and get new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random batch from D.
    Set $z_i = r_i + \gamma\ \text{argmax}_{\hat{a}} \hat{Q}(s_{i+1}, \hat{a_i}; \hat{\theta})$
    Minimize $(z_i - Q(s_i, a_i; \theta))$ w.r.t. θ.
    $\hat{\theta} \leftarrow \theta$ every N steps.
  end for
end for

V. Proposed Deep Reinforcement Learning Approach for Monopoly

The rule-based agents described in the previous section are fixed-policy agents. We propose a model-free agent that is capable of learning an optimal policy based on its interactions with the Monopoly environment. For this purpose, we adopt a deep reinforcement learning (DRL) approach: Deep Q-Network (DQN) [15] trained using experience replay [16]. The agent, instead of random exploration, starts off by imitating one of the strong rule-based agents, in order to guarantee a good initialization and a faster convergence to an optimal policy. As the learning proceeds and the agent learns successful actions, its exploration rate decreases in favor of more exploitation of what it has learnt. This results in a hybrid agent that not only utilizes its own experience memory but also leverages the intelligent behavior of

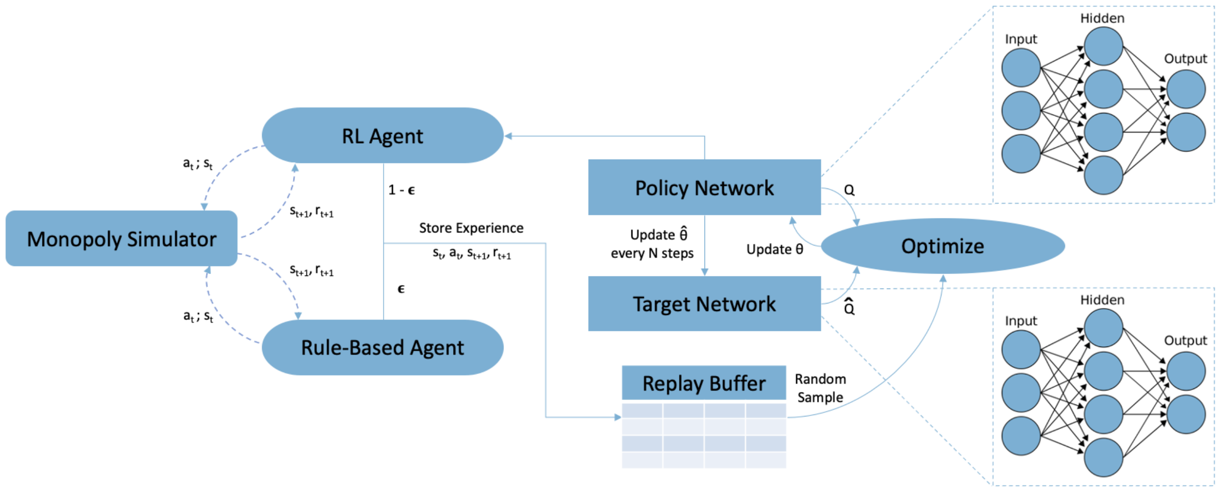

Fig. 2: Deep Reinforcement Learning Approach for Monopoly

a smart rule-based agent that mimics human logic. The overall flow of our approach is presented in Fig. 2.

At each time-step t, the DRL agent selects an action $a_t \in A(s_t)$ based on the current state of the environment $s_t \in S$, where S is the set of possible states and $A(s_t)$ is the finite set of possible actions in state s_t. Similar to [15], we make use of the ϵ-greedy exploration policy to select actions. However, instead of selecting random actions during exploration, we utilize smart rule-based agents to aid exploration. This results in our agent being more decisive than the dithering behavior common to random exploration. The agent imitates the policy of a strong rule-based agent to select an action with probability $\epsilon \in [0, 1]$ and the policy network to select an action otherwise, i.e., during exploitation. This is further explained in Section VI-A.

After an action is executed, the agent receives a reward, $r_t \in R$, and state of the environment is updated to s_{t+1}. The transitions of the form (s_t, a_t, r_t, s_{t+1}) are stored in a cyclic buffer, known as the *replay buffer*. This buffer enables the agent to randomly sample from and train on prior observations. We make use of a target network to calculate the temporal difference error. The target network parameters $\hat{\theta}$ are set to the policy network parameters θ every fixed number of steps. The procedure is described in Algorithm 1. Each episode represents a complete game and each time-step is every instance that the DRL agent needs to take an action within the game. Below, we describe the state space, action space, and reward function of our DRL agent.

A. State Space

We base the state space on the one used in [8]. The state vector contains three different parts:

1) Owned Properties
2) Current Position
3) Financial State

The properties are divided into 10 groups, 8 Monopoly color groups, railroad group, and utilities group. To represent the properties, we use a 10x2 matrix, where the first column contains the percentage of the group owned by the current player and the second column contains the percentage of the group owned by all the other players combined.

The current position of the player is represented relative to a property group. The value ranges from 0 to 1 depending on the property group, the agent is on. For example, if the agent is on an orange property, which is at index 3, the position value would be 3 / (10-1) = 0.33. Similarly, this value would be 1, if the agent is on a utility group (index = 9).

The financial state of the current player has two parts: 1) Current Cash, and 2) Owned Assets. Since the cash value can be either positive or negative throughout the game, we represent the current cash of a player using a Sigmoid function: $\frac{current\ cash}{1+|current\ cash|}$. We represent owned assets as a ratio of the number of assets owned by the current player to that of the number of assets owned by other players. This value becomes 1, when the current player owns all the assets.

We get a 23-dimensional vector to represent the state space – 20 values representing the properties, 1 for the position information, and 2 values for the financial state representation.

B. Action Space

To constrain the action space for Monopoly, we break down the actions into two groups - 1) Property-group actions, and 2) Non-group actions. As the name suggests, a property group action is associated with a property, for example, buy, sell, mortgage, etc. A non-group action does not have any property associated with it, for

Property-group Actions	Non-group actions
Free property	Conclude actions
Mortgage property	Skip turn
Sell property	Pay jail fine
Buy property	Use 'Get out of jail' card
Improve property	Accept trade offer
Make a trade offer	

TABLE I: Possible actions in the Monopoly Simulator

example, skipping a turn, getting out of jail, etc. We have eleven distinct actions, six of which require a property group. The set of possible actions is given in table I.

A property group comprises of 8 color groups (22 "Real Estate" locations), 4 railroads, and 2 utilities. Out of the property group actions, "improve property" is associated with only Real Estate locations, the rest are applicable to any property. The action space is represented as a 63-dimensional vector - 8 values representing the "improve property" action, 50 values representing the 5 remaining actions associated with the 10 property groups, and 5 non-group actions.

Each property group action (except "Buy Property") has multiple parameters that need to be specified. For "Buy Property" action: a player is only allowed to buy a property if it is owned by the bank. When the player lands on a such a property, they have the option of buying it. Note that "asset" and "property" are being used interchangeably in this paper. Once the DRL agent selects an action, a_t, we use the following auxiliary functions to map parameters to the selected action:

Free mortgage: To determine which asset to free from mortgage, this function returns the asset that has the highest mortgage.

Mortgage property: Out of all un-mortgaged assets of the player, this function returns the asset that has the lowest mortgage value to be mortgaged.

Sell property: To decide on which property to sell, this function returns, in order of preference, a hotel, a house or the property itself as the parameter.

Improve property: A property can only be improved, if it is an un-mortgaged "Real Estate" and no hotel exists on it. If improvement is possible and the current player has enough money to build a new house (or a hotel), then house (or hotel) is returned by the mapping function as a parameter for this action.

Make Trade Offer: The purpose of this action is to make an exchange trade offer to other players. To determine which player to trade with, and which asset to offer, we create a priority list. This list consists of all possible trade combinations of the assets of each active player in the game and the current player's assets. The highest priority is given to the assets that allow the current player to acquire a monopoly (own all assets in a color group) and the lowest priority is given to assets that would allow an opponent to monopolize a color group. Otherwise, the priority is determined by the price of the respective properties.

We note that the other actions in Table I have clear meanings since there is only one option to skip turn, pay jail fine, etc., and a specific modification on choosing such an action is not needed, and thus the details are omitted. We note that our choice of action space helps reduce the overall complexity; since without grouping of actions, the overall action space would become much larger limiting efficient training of a learning algorithm.

C. Reward Function

We use a combination of a dense and a sparse reward function (Eq. (1)). In order to reward/penalize a player for the overall policy at the end of each game, we use a constant value of ± 10 for a win/loss respectively.

$$r = \begin{cases} +10 & \text{for a win} \\ -10 & \text{for a loss} \\ r_x & \text{if the game is not over} \end{cases} \tag{1}$$

where r_x is the in-game reward for player x.
During a single game, we use a reward function (Eq. (3)) defined as the ratio of the current players' net-worth (Eq. (2)) to the sum of the net-worth of other active players. The net worth of each active player is calculated after the agent takes an action. This reward value is bounded between [0,1] and helps in distinguishing the relative value of each state-action pair within a game.

$$nw_x = c_x + \sum_{a \in A_x} p_a \tag{2}$$

where nw_x is the net worth of player x, c_x is the current cash with player x, p_a is the price of asset a and A_x is the set of assets owned by player x.

$$r_x = \frac{nw_x}{\sum_{y \in X_i \setminus x} nw_y} \tag{3}$$

where r_x is the in-game reward for player x and X is the set of all active players.

D. DQN Architecture and Parameters

We use a fully connected feedforward network to approximate $Q(s_t, a_t)$ for the policy network. The input to the network is the current state of the environment, s_t, represented as a 23-dimensional vector as seen in the previous section. We make use of 2 hidden layers, that consist 1024 and 512 neurons respectively, each with a rectified linear unit (ReLU) as the activation function:

$$f(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The output layer has a dimension of 63, where each element represents the Q-value for each of the actions the agent can take. For training the network, we use

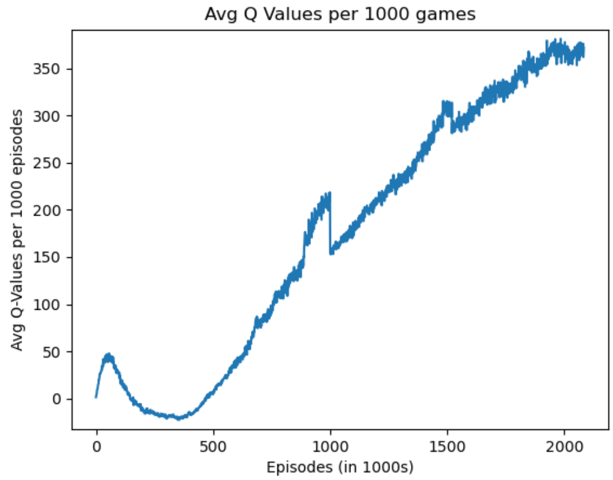

Fig. 3: Average Q Values during Training

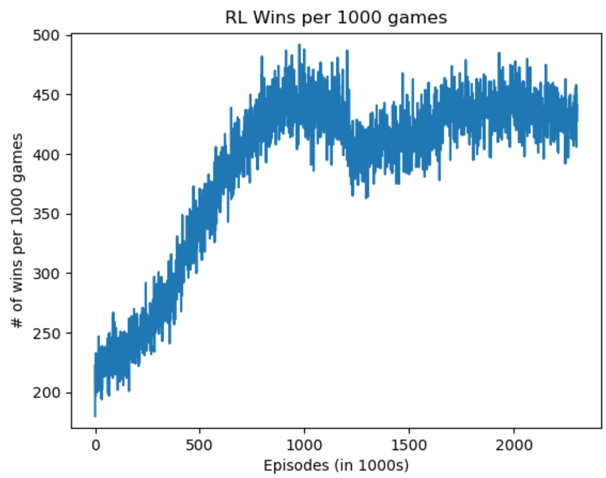

Fig. 4: Win Rate of DRL agent during Training

the Adam optimizer [17] and mean-square error as the loss function. As stated earlier, we initialize the target network with the same architecture and parameters as the policy network. The parameters of the target network are updated to that of the policy network every 1000 episodes and kept constant otherwise. We tuned the DQN, and achieved the best results using the following parameters: $\gamma = 0.99$, learning rate $\alpha = 10^{-3}$, batch size $= 128$, and a memory size $= 10^5$.

VI. Experimental Results

A. DQN Training and Testing

Training: We develop a Monopoly game simulator using Python and Pytorch, available on GitHub[4]. Our simulator enforces all the game rules explained in Section III. We, then, train our RL agent using Deep Q-learning for 2.3 million episodes, where we set 3 players adopting the policy of the smarter rule-based agent (described in Section IV-C) as our opponents. The training for our DRL agent starts with $\epsilon_0 = 1$, in t^{th} game for the player, we choose $\epsilon_t = .01 + .99 \times \exp(-10^{-6}t)$. Unlike the common random ϵ - greedy exploration, our DRL agent utilizes the policies of strong rule-based agents to achieve a strong initialization of its policy.

In order to train the policy, we used a smarter rule-based agent (described in Section IV-C) for the exploration in the first 1.2 million steps (i.e., episodes). After 1.2 million steps, we observe that the win rate of our agent starts to decline as shown in Fig. 4. This decline is attributed to the fierce competition between all four players, following the same policy of the smarter rule-based agent (in Section IV-C) to take ownership of the same set of assets. However, if a player learns this and instead competes for a different set of properties, it can still outperform the other players. To mitigate this issue, for the next 1.1 million steps, our DRL agent sets $\epsilon_t = .01 + .49 \times \exp(-10^{-6}t)$ for game $t > 1.2M$, and starts to imitate the sophisticated rule-based agent (described in Section IV-B) by following its policy to select an action with probability ϵ_t, and the policy network to select an action with probability 1-ϵ_t. This significantly improves the performance of our agent against the other three players (which could adopt the same or different policies) as will be shown through experimental results in sections VI-B and VI-C.

Figure 3 shows the convergence of Q-values at around 1.8 million steps, whereas Fig. 4 shows how the winning rate of our agent improves as learning proceeds. We can observe that our agent recovers from the decline that occurs between 1 and 1.2 million steps, and starts to show a consistent increase in both the average Q-values and the win rate until it converges at around 1.8 million episodes (i.e., games).

Testing: For testing, we use our pre-trained agent in each of the evaluation settings described in Section VI-B. During testing, our hybrid DRL agent not only exploits its learnt policy, but it also imitates the sophisticated rule-based agent (in Section IV-B) with probability 0.2. This hybrid execution results in improved performance for our DRL agent, especially in more competitive settings with multiple sophisticated-policy (Section IV-B) or smarter-policy (Section IV-C) opponents.

B. Baselines and Evaluation Metrics

For evaluations, our Hybrid DRL agent plays against three other players, who potentially adopt different policies, for 50 tournaments of 50 games each. We compare our Hybrid Deep Reinforcement Learning approach against three players of different combinations from the baselines explained below:

1) **Standard DQN Player:** that follows an ϵ-greedy exploration policy to select actions during training,

[4]https://github.com/mayankkejriwal/GNOME-p3

		Basic DQN (P_1)		Simple Baseline (P_2)		Sophisticated Baseline (P_3)		Smarter Baseline (P_4)		Hybrid DRL (Ours)	
Opponent Types	Evaluation Settings	Game Win Rate	Tour-nament Win Rate	Game Win Rate	Tour-nament Win Rate	Game Win Rate	Tour-nament Win Rate	Game Win Rate	Tour-nament Win Rate	Game Win Rate	Tournament Win Rate
Homogeneous Opponents	3 P_1s	24.2	24	42.7	94	**45**	**100**	42.6	100	**54.5**	**100**
	3 P_2s	17.5	4	24	20	**58.4**	**100**	54	100	**73**	**100**
	3 P_3s	3.4	0	9	0	25.5	**24**	**25.8**	22	**41**	**88**
	3 P_4s	1.2	0	10	0	**26**	**22**	25.5	20	**40**	**84**
Heterogeneous Opponents	2 P_1s + 1 P_2	19.1	0	33.7	46	**57**	**100**	52.4	100	**66**	**100**
	2 P_1s + 1 P_3	16.5	2	22.4	1	42.6	40	**46.5**	**68**	**53**	**90**
	2 P_1s + 1 P_4	17.5	4	25	2	39.3	24	**42.2**	**48**	**57.3**	**96**
	1 P_1 + 2 P_2s	17	4	28	28	**60.8**	**100**	55	100	**71.3**	**100**
	1 P_1 + 2 P_3s	7.5	0	12	0	33	38	**45.2**	**91**	**47**	**98**
	1 P_1 + 2 P_4s	5.1	0	15.2	0	28.4	14	**33**	**32**	**48.2**	**100**
	1 P_1 + 1 P_2 + 1 P_3	8.9	0	17.3	0	41.8	54	**44**	**70**	**55.3**	**94**
	1 P_1 + 1 P_2 + 1 P_4	10.9	0	17.2	0	37.6	20	**41.5**	**70**	**56.4**	**98**
	1 P_1 + 1 P_3 + 1 P_4	7.6	0	13.3	0	26	5	**35**	**52**	**44**	**86**
	2 P_2s + 1 P_3	6.5	0	13.6	0	**36.5**	**42**	33.5	22	**53**	**94**
	2 P_2s + 1 P_4s	7.4	0	16.7	0	**41.6**	**72**	36.6	30	**56**	**100**
	1 P_2 + 1 P_3 + 1 P_4	2.8	0	10.8	0	**31.6**	**40**	28	18	**43.5**	**86**
	2 P_3s + 1 P_2	3.3	0	9.6	0	**29**	**32**	26.7	21	**44.5**	**92**
	2 P_3s + 1 P_4	1.2	0	9.2	0	23.9	18	**24**	**19**	**35**	**38**
	2 P_4s + 1 P_2	3.3	0	11.2	0	**33.3**	**42**	29.4	22	**42**	**84**
	2 P_4s + 1 P_3	2.3	0	9.3	0	**24.5**	**26**	24	21	**37**	**76**

TABLE II: Evaluation Settings with homogeneous and heterogeneous opponent types and the corresponding win rates (%) of four baselines Vs. our proposed Hybrid DRL agent

where it picks randomly one of the allowable actions to perform at time step τ with probability $\epsilon \in [0, 1]$ and exploits its experience memory and its policy network to select an action with probability $1 - \epsilon$. However, it does not adopt any imitation learning, it starts learning from scratch without initializing its strategy. We have trained this agent for 1.5 million episodes. We will denote this player P_1 henceforth.

2) **Simple rule-based player:** that is explained in Section IV-A, will be denoted P_2 henceforth.
3) **Sophisticated rule-based player:** that adopts additional human logic in handling more complex situations during the game, explained in Section IV-B and will be denoted P_3 henceforth.
4) **Smarter rule-based player:** that adopts the complex logic of the P_3 in addition to aggressively bidding and targeting more expensive assets to own, as explained in Section IV-C. We denote this player P_4.

Table II shows the different settings (i.e., compositions of three opponents) utilized for evaluations against our DRL player. For each one of these settings, we calculated the average win rate of our Hybrid DRL agent over 50 tournaments, each of 50 games. We investigate both the game win rate (percentage of game-wins out of all 2500 games), and the tournament win rate (percentage of tournament-wins out of all 50 tournaments). Note that, the player with the most game wins in a given tournament is considered the tournament winner, and ties are not counted towards the tournament win rate.

We hypothesize that our DRL agent will be able to beat all settings that are composed only of P_1 and P_2 opponents by a large margin, due to learning the strong policy of the smarter rule-based agent (in Section IV-C). In addition, it will also outperform in settings that involve several P_3 opponents as it is trained to beat the smarter rule-based agent. However, it becomes more challenging when there is more than one P_4 in the opponents, as their policies start to fire back on each other as they compete for the same set of properties. In that case, a strong rule-based agent that might not be as smart can win over. We hypothesize that our hybrid DRL will still outperform in this case, because it not only exploits its learnt policy that started off by imitating the smarter rule-based agent (in Section IV-C, but it also utilizes the policy of the sophisticated rule-based agent (in Section IV-B). We will also show that even in the most competitive settings where the opponents are composed of strong players (i.e., P_3s and P_4), our DRL agent outperforms by a significantly large margin in both performance measures.

C. Results Discussion

We test our proposed DRL agent against *homogeneous opponents*, where all the opponents adopt the same policy for playing; as well as against *heterogeneous opponents*, where the opponents adopt varying policies. From our simulation, we observe that the hypothesis for each baseline comparison has been supported for the most part by our experimental results. In Table II, we investigate the performance of our proposed framework in comparison to all other baselines, where we show the game and tournament win rates of each of the four baselines as well as those of our proposed hybrid DRL agent. The highest game win rate among all baselines is highlighted in **purple**, while the highest tournament win rate is highlighted in **violet**.

We can observe that in homogeneous settings, both P_3 and P_4 wins against all 3 P_3s or P_4s by $\approx 25\%$; while our proposed DRL agent exceeds by 15% to win $\approx 40\%$ of the games and achieves 4 times their tournament win rate reaching $> 80\%$. For heterogeneous settings, we conclude that the smarter baseline P_4 is able to outperform other baselines in almost all settings that has ≤ 1 P_3 or P_4 opponent players. However, when this increases to involve 2 or more P_3s or P_4s, the sophisticated baseline P_3 tends to outperform the other baseline. This is attributed to the fact that the higher the number of players adopting the same policy, the more it becomes a disadvantage to all of them as the competition becomes more fierce on the set of properties they are targeting.

However, our hybrid DRL approach is able to achieve the balance, and performs winning actions in all settings that enable it to outperform all baselines. The hybrid execution of our agent as it exploits its learnt policy with probability 0.8, and utilizes the policy of the sophisticated rule-based agent with probability 0.2, allows it to reach this balance. Our experimental results support this hypothesis, as we can observe that our approach exceeds the best baseline performance in the game win rate by at least a margin of $15-25\%$, and $> 40\%$ in the tournament win rate in all the previously mentioned settings. In addition, in the completely heterogeneous settings, where none of the opponents adopt the same policy, our proposed hybrid agent excels against all other baselines, by showing a 10% increase in the game win rate, and $> 25\%$ increase in the tournament win rate over the best baseline results. Besides, in the most competitive settings, where the set of opponents is composed entirely of strong agents (i.e., P_3 and P_4 only), our proposed baseline shows an improvement of $10-15\%$ in the game win rate, and $20-50\%$ in the tournament win rate.

VII. Conclusion

In this paper, we utilize a deep reinforcement learning (DRL) approach to model Monopoly as an MDP. Our hybrid DRL approach starts its learning procedure by imitating a strong rule-based agent (that mimics human logic) to initialize its policy and to obtain decisive options that aid its exploration. Then, our agent improves its policy using deep Q-learning that converges, after around 1.8 million episodes, to a winning strategy. In addition, we propose three strong rule-based agents that reflect successful strategies adopted by actual human players. Finally, our evaluation results show the effectiveness of our approach in outperforming all baselines, in both homogeneous and heterogeneous settings, by a large margin (above 15% in the game win rate, and above 40% in the tournament win rate). Extension of this work to train multiple agents using Multi-Agent Reinforcement Learning (MARL) is left as future work.

References

[1] A. Celli, A. Marchesi, T. Bianchi, and N. Gatti, "Learning to correlate in multi-player general-sum sequential games," in *Advances in Neural Information Processing Systems*, 2019, pp. 13 076–13 086.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, "Creating pro-level ai for a real-time fighting game using deep reinforcement learning," *IEEE Transactions on Games*, 2021.

[4] P. Paquette, Y. Lu, S. S. Bocco, M. Smith, O. G. Satya, J. K. Kummerfeld, J. Pineau, S. Singh, and A. C. Courville, "No press diplomacy: Modeling multi agent gameplay," in *Advances in Neural Information Processing Systems*, 2019, p. 4474 4485.

[5] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert level artificial intelligence in heads up no limit poker," *Science*, vol. 356, no. 6337, p. 508 513, 2017.

[6] N. Brown and T. Sandholm, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, p. 885 890, 2019.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi agent reinforcement learning," *Nature*, vol. 575, no. 7782, p. 350 354, 2019.

[8] P. Bailis, A. Fachantidis, and I. Vlahavas, "Learning to play monopoly: A reinforcement learning approach," in *Proceedings of the 50th Anniversary Convention of The Society for the Study of Artificial Intelligence and Simulation of Behaviour. AISB*, 2014.

[9] E. Arun, H. Rajesh, D. Chakrabarti, H. Cherala, and K. George, "Monopoly using reinforcement learning," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 858–862.

[10] F. Cacciamani, "State of the art on: Efficient solutions for adversarial team games," 2019.

[11] A. Celli, M. Ciccone, R. Bongo, and N. Gatti, "Coordination in adversarial sequential team games via multi-agent deep reinforcement learning," *arXiv preprint arXiv:1912.07712*, 2019.

[12] S. Sammul, "Learning to play monopoly with monte carlo tree search," *Computer science bachelor thesis, School of Informatics, University of Edinburgh*, 2018.

[13] R. B. Ash and R. L. Bishop, "Monopoly as a markov process," *Mathematics Magazine*, vol. 45, no. 1, pp. 26–29, 1972.

[14] M. S. Dobre and A. Lascarides, "Exploiting action categories in learning complex games," in *2017 Intelligent Systems Conference (IntelliSys)*. IEEE, 2017, pp. 729–737.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

APPENDIX

A. General Rules

- During the start of the game, each player gets $1500 in cash while the Bank holds on to the remaining cash and ownership of properties. The player must roll a pair of dice during its turn and move forward by the sum of the number on the dice.
- If the player lands on an un-owned property, the player may buy it for the price listed on that property's space. If he/she agrees to buy it, he/she pays the Bank that amount and receives the deed for that property. If he/she refuses to buy the property for the amount stated, the property is auctioned. Players must bid an amount greater than the existing bid in order to remain in the in the auction. The player that bid the highest amount wins the auction. Railroads and utilities are also considered properties.
- If the player lands on an un-mortgaged property owned by another player, he/she pays rent to that person, as specified on the property's deed.
- If the player lands on the Luxury Tax or Income Tax locations, it must pay an the stated tax amount to the Bank.
- If the player lands on a Chance or Community Chest, the player takes a card from the top of the respective pack and performs the instruction given on the card.
- The player may sell property and improvements (houses and hotels) back to the Bank at half the purchase price
- In this paper, doubles do not receive any special treatment.
- When in jail, a player may use the "get out of jail" free card or pay the $50 jail fine or can skip one turn. Players in jail are still allowed to collect rent and perform other *out of turn and pre-roll* actions.

B. Construction Rules

- Once a player owns all properties of a colour group, i.e., it has acquired a *"Monopoly"*, the rent on all unimproved properties of that color group are doubled, even if any of the properties in that color group are mortgaged.
- The upper limit for the maximum number of houses available with the bank for purchase during a game is 32. If all the 32 houses are bought by the players, then no more houses can be purchased from the bank until one or more houses are returned to the bank. The upper limit for the maximum number of hotels available with the bank for purchase during a game is 12.
- Properties must follow the uniform improvement rule, i.e., each house must be built on the property with the least number of houses in that color group. A hotel may be setup only after all the properties in that color group have been improved with four houses each. A hotel can be setup by paying the price of an additional house and by returning the four houses back to the Bank.
- At any time a player may, to raise cash, sell hotels and houses back to the Bank for half the purchase price of the houses or hotels. Also, properties with no houses or hotels may be mortgaged for half of the property price. A property does not collect rent while mortgaged and may not be developed. To de-mortgage a property a player must pay interest of 10% in addition to the mortgage price. Whenever a mortgaged property changes hands between players, either through a trade, sale or by bankruptcy, the new owner must immediately pay 10% interest on the mortgage and at their option may pay the principal or hold the property. If the player holds the property and later wishes to lift the mortgage they must pay an additional 10% interest at that time.
- **RailRoads:** The rent a player charges for landing on a railroad varies with the number of railroads that are also owned by a player. The rent is as follows: Charge $25 if one is owned, $50 if two are owned, $100 if three are owned, $200 if all four are owned.
- **Utilities:** After a player lands on one utility to owe rent, the rent is 4 times the amount rolled, if the other player owns one utility. If the other player possesses both utilities, the rent is 10 times the amount rolled.