Received 28 May 2023, accepted 11 June 2023, date of publication 16 June 2023, date of current version 21 June 2023.
Digital Object Identifier 10.1109/ACCESS.2023.3287100

RESEARCH ARTICLE

Reinforcement Learning Agents Playing Ticket to Ride–A Complex Imperfect Information Board Game With Delayed Rewards

SHUO YANG[1], MICHAEL BARLOW[1], THOMAS TOWNSEND[2], XUEJIE LIU[1], DILINI SAMARASINGHE[1], ERANDI LAKSHIKA[1], (Senior Member, IEEE), GLENNN MOY[3], TIMOTHY LYNAR[1], AND BENJAMIN TURNBULL[1], (Senior Member, IEEE)
[1]School of Engineering and Information Technology, University of New South Wales, Canberra, ACT 2600, Australia
[2]School of Professional Studies, University of New South Wales, Canberra, ACT 2600, Australia
[3]Defence Science and Technology Group, Edinburgh, SA 5111, Australia

Corresponding author: Shuo Yang (shuo.yang5@adfa.edu.au)

This work was supported in part by the Defence Science and Technology Group (DSTG) of the Department of Defence, Australia, through the Australian Department of Defence's "Defence Science Partnerships (DSP) Multi-Party Collaborative Project Agreement," under Agreement 10923.

ABSTRACT Board games are extensively studied in the AI community because of their ability to reflect/represent real-world problems with a high-level of abstraction, and their irreplaceable role as testbeds of state-of-the-art AI algorithms. Modern board games are commonly featured with partially observable state spaces and imperfect information. Despite some recent successes in AI tackling perfect information board games like chess and Go, most imperfect information games are still challenging and have yet to be solved. This paper empirically explores the capabilities of a state-of-the-art Reinforcement Learning (RL) algorithm – Proximal Policy Optimization (PPO) in playing Ticket to Ride, a popular board game with features of imperfect information, large state-action space, and delayed rewards. This paper explores the feasibility of the proposed generalizable modelling and training schemes using a general-purpose RL algorithm with no domain knowledge-based heuristics beyond game rules, game states and scores to tackle this complex imperfect information game. The performance of the proposed methodology is demonstrated in a scaled-down version of Ticket to Ride with a range of RL agents obtained with different training schemes. All RL agents achieve clear advantages over a set of well-designed heuristic agents. The agent constructed through a self-play training scheme outperforms the other RL agents in a Round Robin tournament. The high performance and versality of this self-play agent provide a solid demonstration of the capabilities of this framework.

INDEX TERMS Board games, imperfect information games, proximal policy optimization, reinforcement learning, ticket to ride.

I. INTRODUCTION

Compared to other forms of games (e.g., video games, sports, and card games), board games can be high-level abstractions of complex problems including real-world problems [1], [2]. Some board games are designed with a purpose to provide an abstract version of real-world mechanisms or problem spaces. For example, the game QE models the economic mechanism of Quantitative Easing [3], while the COIN series games model asymmetric historical conflicts [4]. Some board games are constructed to highlight insights about real-world domains, analyze historic events in greater depth or otherwise inspire understanding and knowledge of the real-world contexts represented. Examples of these are the games Cuba Libre [5], and Xiang Qi [6]. In short, beyond entertainment, board games often have practical significance. One significant example of board games impacting real-world decision-making is in wargaming, which has been used

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaogang Jin.

throughout history to aid decision makers [7], [8]. Therefore, board games can be highly suitable vehicles for research. Board games are also important benchmark problems of artificial intelligence research [9], [10], [11], [12]. Many key moments in the evolution of Artificial Intelligence (AI) have been introduced through the implementation of AI playing board games. Examples of this include Deep Blue becoming the first computer program to defeat a human chess champion in 1997 [13], AlphaGo [14] defeating human Go masters in 2016, given that Go was previously considered the most challenging classic games for AI [15], and its successor, AlphaZero [16], achieving even better results without relying on human data, guidance or domain knowledge beyond game rules.

This paper investigates the ability of a general-purpose Reinforcement Learning (RL) algorithm - Proximal Policy Optimization (PPO) - to learn to play the commercial board game "Ticket to Ride" (T2R). T2R is a world-famous network building game with imperfect information and delayed rewards [17]. Winning the game requires considerable planning on resources acquisition, network building and contract fulfilment. Reaching a high level of performance in T2R is challenging for current AI algorithms [17]. There are several reasons for this. Firstly, the state-action space of T2R is huge, even compared to games such as chess. The original version of T2R has more than 10^{54} possible states, which is at least 10^{11} higher than that of chess [18]. The action space is also large given the various options provided for a player, which culminates with a high branching factor for a tree-based algorithm. Secondly, T2R is also an imperfect information game with stochasticity as it has card drafting and hand management mechanisms like poker does, which makes the game non-deterministic. Finally, as delayed reward structures are challenging for RL agents [19], [20], [21], each delayed reward involved in fulfilling a hidden contract adds extra difficulty in converging to the optimal value function/network and the number of contracts is even indefinite. Because of these difficulties, existing T2R AI agents have either not demonstrated human level abilities or contain a considerable proportion of rule-based heuristics [17], [18], [22].

Being able to solve a complex imperfect information game with a general-purpose algorithm is of great importance [23], [24]. Firstly, although state-of-the-art algorithms such as tree-based, game theory-based, Counterfactual Regret Minimization (CFR) algorithms, and/or RL based algorithms have achieved superhuman performance in some imperfect information game with smaller state-action space [9], [12], [25], [26], [27], [28], [29], complex imperfect information games remain an open challenge for AI. Secondly, to achieve superhuman performance in more complex board games, existing algorithms usually require the usage of human data, guidance or domain knowledge in the architecture design, feature selection, and fine tuning of task specific hyperparameters [16], [30].

This paper demonstrates the feasibility of using a general-purpose RL algorithm to play T2R with no domain knowledge beyond game rules, game states and scores. Inductive biases are only introduced when necessary, such as the reward shaping in line with the game rules and the intuitive ways of encoding the state-action space. In a two-player setting, different training strategies with different opponent arrangements are introduced in the training process including a self-play strategy [31]. Considering the high complexity of public versions of T2R, a simplified version of T2R is created to limit the computational requirements of the early experiments. The performance of the RL agents is tested on a variety of well-designed heuristic AI players [32].

The contributions of this paper include:

1) The imperfect information game T2R is formalized into a Partially Observable Markov Decision Process (POMDP). The modelling approaches can be generalized to other complex board games with similar settings when applying general purpose RL algorithms.
2) A range of RL agents are obtained through different RL training schemes. These RL agents are demonstrated to achieve advantages over a range of well-designed heuristic agents in the simplified version of T2R.
3) The capability of a general-purpose RL algorithm (PPO) to play a complex, imperfect information board game (T2R) with limited domain knowledge is demonstrated. A self-play agent that more strictly excludes the impact of domain knowledge outperforms all the other RL agents and is demonstrated to be more versatile in competing with different opponents including both heuristics and RL agents.

The remainder of this paper is as follows. Firstly, the background is given in Section II, including an introduction to T2R, and the current development of AI agents in playing board games. Section III details the proposed methodology: Section III-A models T2R with detailed descriptions of the state space and action space; Section III-B gives the configuration of the scaled-down map; Section III-C summarizes the strategies of the heuristic agents; Section III-D specifies the training schemes for obtaining RL agents. All experimental results are provided in Section IV including the performance of the RL agents evaluated through games with the heuristic agents and a Round Robin tournament among the RL agents. Section V summarizes this paper.

II. BACKGROUND

A. TICKET TO RIDE

Ticket to Ride (T2R) is a competitive, mid-weight (complexity) Euro game [33] in which the players are budding tycoons seeking to lay their rail networks across a continent/country/region; to maximize the routes they cover and most importantly satisfy individual contracts or destination tickets (varied cities on the map that they must connect through their rail networks) [34].

There are three types of actions available at each game turn (as in the original version of T2R). The first action is edge connection. The board begins with a pre-defined network between cities and without any established networks. These networks emerge through play and are constructed an edge at a time by the players. Each edge requires a certain number and type(s) of resource to be built, with the number representing the length (or span) of that connection and the type being specified by a single color. These individual resources are represented as train cards of different colors (red, yellow, blue, black, etc.). Each connection of an edge must be built in a single action, and only after the player has gathered and then spent, the necessary resources (i.e., a certain number of train cards with the right color). At the same time, the equivalent number of player tokens (plastic train cars in the physical board game) are deployed on the map as a representation of occupying that edge. The second action is drawing train cards. The train cards are collected by players through a card drafting mechanism, which provides players the options to draw from a stack of face-down cards, or from 5 face-up cards. The third action is drawing destination tickets (destination tickets are used interchangeably with tickets throughout this paper). As another non-trivial winning strategy, drawing tickets comes with high stake as completing a ticket (connecting the two destinations named on the ticket through building successive connections) results in bonus points (written on the ticket, and proportional to the difficulty of its fulfilment), but failing to do so results in deduction of the same number of points. It is mandatory to keep a minimal number of destination tickets at the start of a game and more can be drawn during the game. Ticket points are finalized at the end of a game.

The end-game condition is triggered once the usage of tokens of a player has passed a threshold. The winner is determined by calculating the total points of the players at the end of the game.

T2R is a partially observable game. The cards of a player are hidden from the others including tickets and train cards. Tickets are always hidden from the other players. Although the train card drawn from the face-up cards are known by the other players, the initial cards and the cards drawn from stack are unknown. Stochasticity is introduced by the drafting mechanisms of tickets and train cards.

B. AI IN BOARD GAMES

1) HEURISTIC AGENTS

Heuristic agents rely on heuristic rules and evaluations that are abstracted from a priori knowledge of a pre-defined problem. Before the rise of computationally intensive Artificial Intelligence (AI), heuristic agents had been the most intuitive and feasible way to build an AI agent in playing games [35], [36].

Pure heuristic agents have advantages and disadvantages. A heuristic agent can directly inherit human knowledge in playing the game without a learning process, although the abstraction and modelling of human knowledge can be a challenge. Consequently, it does not rely on a large amount of game data to form its policy, and thus, is less computationally demanding. However, heuristic agents tend to be inflexible and predictable in their moves, aiding in the development of effective counterstrategies [37]. Heuristics also typically need to be re-developed for new game environments, making them less useful as general game-playing algorithms. Due to the pre-defined nature, a heuristic agent may not be versatile in all circumstances so that its performance may be sensitive to circumstance changes. This opens the opportunity to build heuristics with different characteristics [17], [38]. Silva et al. [17] developed four heuristic agents playing T2R with different playing styles. The heuristic agents with different "characteristics" were inspired by the game strategies shared in the community of T2R. Such a variety set of players were used for automated playtesting. Two classes of failure states were identified. These heuristic agents were used in analyzing the characteristics of different maps including the most favorable cities.

Heuristic rules and evaluation methods are still important parts of state-of-the-art AI players since they are commonly integrated into modern AI algorithms such as game theory-based, tree-based approaches, Monte Carlo Tree Search, and Deep Reinforcement Learning [12], [13], [20], [37], [39] to accelerate the learning process of the combined policy. Beyond that, heuristic agents are often used as benchmarks thanks to their ease of implementation and limited demand in computational power [38].

2) SEARCH ALGORITHMS

Tree-search algorithms are commonly used to solve problems with sequential decisions, including games [40], [41]. For a game of perfect information, a comprehensive search of the tree with a proper evaluation method can be used to determine the optimal moves. The size of a search tree is determined by its width (number of legal moves per position) and depth (number of turns from a position to a final game state) [14]. For complex games with large search trees (e.g., chess and Go), it is usually impractical to conduct an exhaustive search. Methods have been developed to reduce the search space by reducing the depth [42], [43], [44] and the width of the search [13], [45], [46]. Monte Carlo Tree Search (MCTS), and its variants, have been extensively used in solving games with such large search trees.

MCTS has shown advantages in large, fully observable games [47], [48], [49], [50], such as chess [51] and Go [52], [53]. However, it is a more normal case in games to have partly observable states from the perspective of one or more players. These games are also referred to as imperfect information games (the opposite to perfect information games) [9]. Imperfect information games can be easily found in card games (e.g., bridge, Dou Di Zhu and Skat), board games (e.g., backgammon, Settlers of Catan and Ticket to Ride), and digital Real-Time Strategy (RTS) games with the common setting of the "fog of war" (e.g., Starcraft II, Dota 2 and League of Legends). Game theory has extensively studied imperfect information games, but the complexity of most real-world games makes it impractical to obtain a Nash Equilibrium strategy as proper abstraction is required, e.g., Counterfactual Regret Minimization (CFR) [54].

Extensions and modifications of MCTS have been developed to address imperfect information games. One approach to the imperfect information problem is *determinization*, also known as Perfect Information Monte Carlo (PIMC) [9], [55]. However, determinization MCTS faces some major challenges including strategy fusion, non-locality, computational budget sharing, and fake omniscience [56], [57]. As a result, determinization MCTS can have poor performance in many real-world games [58].

Information Set MCTS (ISMCTS) was proposed to address the strategy fusion and computational budget problems [38], [57], [59]. With each node representing an information set rather than states, statistics about moves can be recorded in this one tree instead of the multiple trees derived from determinization methods, which is more computationally efficient [57]. Furthermore, ISMCTS addresses the strategy fusion problem by sharing information between determinizations. Huchler [18] developed AI players to play T2R (original version) under the MCTS framework. Four agents were developed including the two variants of ISMCTS (Single-Observer MCTS and Multi-Observer MCTS) [57], determinized UCT and a voting strategy with the options of several classic enhancements. Although the positive impact of some enhancements to MCTS was confirmed, it demonstrated the MCTS agents were not superior to a Flat Monte Carlo agent in the T2R implementation. There is need for improvement for ISMCTS when it deals with large branching factor at opponent nodes [57]. In addition, ISMCTS hinders the searching player to exploit situations where opponent is lack of information, which is the normal case in a real game [57].

Belief Distribution is another approach to tackle imperfect information games. It usually involves the inference of the complete game states based on historic and current game information [60], [61]. Silver and Veness [50] proposed POMCP to achieve online planning in large Partially Observable Markov Processes (POMDPs) through integrating a Monte-Carlo update of the agent's belief state with MCTS. The belief state was approximated using an unweighted particle filter. High level performance was achieved with no prior knowledge in tackling 3 POMDPs (games). Uriarte and Ontañón [39] enhanced MCTS with single belief state generation by proposing three sampling strategies in the context of a partially observable Real Time Strategy game (μRTS). MCTS enhanced by each of these sampling strategies generated superior performance than the baseline performance generated with an ϵ-Greedy MCTS [62].

In addition, opponent modelling is another approach to address imperfect information MCTS as the hidden information in imperfect information games is usually related to the lack of opponent information [63], [64], [65], [66].

3) REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a machine learning algorithm which enables an agent to learn a policy for choosing a sequence of actions which optimizes the long-term total reward achieved when interacting with an environment [67]. RL problems are often formulated as Markov Decision Processes (MDPs), in which the next state of the system is only related to the current state and the action. At a time point t, the agent performs an action a_t given an observation of the environment s_t (aka. a state) through its policy $\pi(a_t|s_t)$, which denotes the possibility of performing action a_t at state s_t. Each time an agent performs an action, the agent receives a reward r_t from the environment and the environment is updated to the next state s_{t+1}. The state transition can be formulated through transition probability $P(s_{t+1}|s_t, a_t)$. The goodness of an action a_t at state s_t using policy π can be determined by an action-value function $Q_\pi(s_t, a_t)$. Furthermore, state-value V_π which determines the value of state s_t considering all possible actions $a_t \in A$ when applying policy π can be denoted as $V_\pi(s_t) = \sum_{a_t \in A} \pi(a_t|s_t) \cdot Q_\pi(s_t, a_t)$. RL algorithms can be categorized as model-based and model-free depending on whether to develop a predictive model to approximate state transition; model-based algorithms obtain transition functions through learning, by contrast, model-free algorithms do not. Within the class of model-free reinforcement learning algorithms, there are two basic approaches to training RL, namely, value-based approaches and policy-based approaches. Value-based RL algorithms learn the optimal value function $Q^*_\pi(s_t, a_t)$ so that the RL agent can choose actions with the highest value. Policy-based algorithms optimize the agent's policy $\pi(a_t|s_t)$ and perform actions based on the optimized policy. Actor-critic algorithms are further developments building on the two basic branches. They combine the learning of the policy and value function. In the training, the actor learns the action policy based on the estimated state value provided by the critic (value function) which is also part of the training loop. In practice, to solve complex problems with a large number of state-action pairs, neural networks, especially deep neural networks, are usually used to approximate value functions and action policies.

RL is renowned for its broad application potential in solving real world problems. This is because RL enables learning through consuming the decision outcomes of the learning agents. A high degree of autonomy bypasses the need for constant expert input into the training [68]. Therefore, RL, especially DRL, is widely used in solving a large range of real-world problems, especially the ones involve sequential decision-making. For example, in robotics, RL has been extensively used in learning tasks in the physical world with practical constraints, such as manipulation [69], grasping [70] and legged locomotion tasks [71]; in recommending systems, RL has been leading recent developments because the recommendation problem can be formulated as a sequential decision problem across the long-term user engagement [72]; in transportation, RL has been applied to both emerging and conventional problems such as autonomous driving [73], energy efficient driving [74], traffic control [75] and vehicle routing [76] in different transportation systems; in manufacturing, RL has showed its efficacy in improving production scheduling [77] and maintenance [78]; in energy, RL is applied to major subfields such as energy management systems [79], dispatch [80], vehicle energy systems [81], [82], [83], grid [84] and energy markets [85]; in gaming, RL has achieved success in reaching super-human performance [11], improving gaming experience [86] and game development [87]. The list goes on. Among these application fields, gaming is one of the major arenas that propel innovation in RL.

One of the earliest successes of applying RL in game playing is the TD-Gammon proposed by Tesauro [88]. Through a self-play strategy with a copy of the RL agent itself, it was able to approximate an evaluation function with little a priori domain knowledge. Augmented with domain knowledge, TD-Gammon surpassed most human grandmasters in backgammon. Pfeiffer [37] synthesized an RL-based AI with a priori knowledge for Settlers of Catan. The Q-function was approximated through a tree-based approximator. A hierarchical RL framework was used with a perspective that the high-level policy determines whether to execute low-level policies. Heuristics based on a priori knowledge were used in different parts of the framework including feature selection and calculation, and low-level action determination.

Deep Q-Network (DQN) uses a deep neural network to learn the optimal value function and has achieved huge success in harnessing high-dimensional sensory inputs for learning action policies [11]. This success has been marked by the universality and the high performance in classic Atari 2600 games using only inputs from pixels from the displays and game scores. In the realm of board games, Xenou et al. [36] proposed a novel online Deep Recurrent RL (DRRL) algorithm by implementing Q-decomposition [89] to achieve local action evaluation only with action-dependent features. DRRL was used in Settlers of Catan and outperformed the state-of-the-art heuristic jSettler [35]. Strömberg and Lind [22] used DQN to tackle T2R in a simplified version of the original version of T2R. Several agents were obtained with different reward systems through RL. The RL agents achieved high win rate ($>80\%$) versus an agent taking completely random actions.

AlphaGo [14] was the first computer program to defeat a human professional player in the game of Go, a game famous for its enormous search space and viewed as a grand challenge for AI algorithms [15]. AlphaGo integrates RL with MCTS by using a policy network (a deep neural network) and a value network (with similar structure as the policy network) for move selection and position evaluation in a Monte Carlo search algorithm. In the pipeline of training AlphaGo, the policy network was initialized through supervised learning using expert human player moves before it was further improved through self-play with randomly selected previous iteration of the policy network. The value network was trained by regression on state-outcome pairs based on 30 million distinct states with each sampled from a separate game. AlphaGo required significant amounts of human expert moves in the training process, to provide domain knowledge. To address this problem, AlphaZero [16] was proposed with several improvements and the most prominent difference is that it depends solely on a self-play reinforcement learning strategy, i.e., training from tabula rasa with no human data. During training from games of self-play, guided by a versatile deep neural network (which combines the roles of the policy network and the value network in AlphaGo), MCTS was used in selecting each move. AlphaZero outperformed the strongest previous versions of AlphaGo. AlphaZero was further generalized to other games including Shogi and chess [51]. However, neither AlphaGo nor AlphaZero was designed for imperfect information games, especially the ones with a high branching factor.

Brown et al. [28] proposed Deep Counterfactual Regret Minimization (Deep CFR) to address the dependence of domain knowledge of conventional tabular CFRs in the abstraction of game states. It was achieved through using function approximation to approximate the behavior of CFR. While it achieved good performance in large poker games (2-player zero-sum imperfect information games), more scalable sampling strategies will be needed to extend Deep CFR to larger games.

Proximal Policy Optimization (PPO) is a state-of-the-art RL algorithm [90]. It was developed by OpenAI as a general-purpose RL algorithm to address the stepsize problems experienced by the existing policy gradient methods. It is well known for its ease of implementation, appropriate sample complexity and ease of tuning [91]. It has been showing comparable or better performance in Atari benchmarks compared to other widely used algorithms such as TRPO [92] and ACER [93]. PPO has boosted the recent success of defeating human world champions in Dota 2 – a popular imperfect information zero-sum competitive game with two opposing teams – through the OpenAI Five AI system [20]. In this PPO powered AI system, self-play was implemented in the policy learning process. Reward shaping was introduced to balance the relative importance between individual reward and team reward. Considering the merits of PPO such as the high compatibility, good performance and ease of use, it is chosen to be the algorithm in our investigation into the AI capabilities in playing T2R.

III. METHODOLOGY

This section models T2R as a Partially Observable Markov Decision Process (POMDP) with detailed descriptions of the state space and action space. Secondly, the scaled-down version of T2R is created based on the design principles

TABLE 1. Nomenclature.

Symbol	Description
ρ	Number of players
n	Number of cities
m	Number of edges (a double edge counts for 2 separate edges, etc.)
C	Total number of train cars (player tokens) for each player
R	Number of colors involved (shown on train cards and edges)
N_C	Number of cards in each color
N_W	Number of locomotive (wild) cards
D	Number of all destination tickets
N_P	Number of delt destination tickets to choose from
$\|\boldsymbol{Z}\|$	Number of possible states in the observable state space
$\|\boldsymbol{Z}_B\|$	Number of all possible states on the board
$\|\boldsymbol{Z}_F\|$	Number of combinations of face-up cards
$\|\boldsymbol{Z}_H\|$	Number of combinations of hand cards
$\|\boldsymbol{Z}_D\|$	Number of states in terms of the destination tickets
$\|\boldsymbol{A}\|$	Number of possible actions in the action space
$\|\boldsymbol{A}_{tr}\|$	Number of possible actions for drawing train cards
$\|\boldsymbol{A}_c\|$	Number of possible actions of connecting cities
$\|\boldsymbol{A}_d\|$	Number of all possible actions of drawing tickets
cc	Number of card choices to complete a connection
ns	Number of cards needed to complete an edge
m_c	Number of colored edges
m_g	Number of no color edges (grey edges)

abstracted from the analysis of different versions of T2R with various complexity (the abstracted design principles can be found in Appendix A). Thirdly, the heuristic agents used in the experimental demonstration are introduced. Finally, the proposed training schemes for obtaining RL agents are specified. The symbols and abbreviations used in this section are summarized in Table 1.

A. MODELLING TICKET TO RIDE

The POMDP of Ticket to Ride (T2R) can be modelled through a tuple G = $(\boldsymbol{P}, \boldsymbol{S}, \boldsymbol{Z}, O, \boldsymbol{A}, L, T, W, s_0)$ [39]. $\boldsymbol{P} = \{p_1, p_2, \ldots, p_\rho\}$ is the set of players. $\boldsymbol{S}$ is the set of possible game states. $\boldsymbol{Z}$ is the set of possible observations. O(p, s) observation function at state s from the point of view of player p. $\boldsymbol{A}$ is the set of possible actions. $L(p, a, s)$ is a function which returns whether action a is a legal action at state s for player p. $T(s_t, a)$ is the transition function. W is the function which determines the winner. s_0 is the initial state.

The modelling of state space and action space for T2R is elaborated in the rest of this section.

1) STATE SPACE

For each player playing T2R, only partial information of the game can be perceived, i.e., POMDP. Due to the nature of the problem, it is not intended to construct a comprehensive state space with complete information. Rather, only the observable states are taken into consideration. This observable state space involves 6 distinct dimensions which are defined by the corresponding variables that are related to the status of the game. The representation and encoding approach of each dimension are explained in detail together with the equations for calculating the number of possible states.

a: BOARD STATES

Board status is composed of a combination of the state of each edge on the board, i.e., which edges are filled, which edges are claimed by each player. It arguably contains the most important information reflecting the state of the game. Each edge has $(\rho + 1)$ mutually exclusive possible states, i.e., an edge is either claimed by Player 1, or Player 2, ..., or Player ρ, or no one, ρ is the number of players. The board status is denoted as $\boldsymbol{Z_B} = \{B_1, B_2, \ldots, B_m\}$, where B_i represents the state of Edge i, the total number of edges is m. By considering the possible states of all edges, an estimate of the number of possible board statuses $|\boldsymbol{Z_B}|$ can be calculated by (1).

$$|\boldsymbol{Z_B}| = (\rho + 1)^m \tag{1}$$

b: TRAIN CARS

The number of train cars (player tokens) is fixed at the beginning of the game. The number used (and remaining) is public information. It is an important indicator of the game state as it is the variable triggering game end conditions. The number of train cars left can be any integer between 0 and C. The state of train cars left is denoted as $\boldsymbol{Z_C} = \{C_1, C_2, \ldots, C_\rho\}$, where C_i is the number of train cars (player tokens) left for Player i. The number of states originated from the number of train cars left $|\boldsymbol{Z_C}|$ can be calculated by (2).

$$|\boldsymbol{Z_C}| = (C + 1)^\rho \tag{2}$$

c: FACE-UP CARDS

As written in the rules of T2R, there are always 5 face-up (train) cards shown as the available options for players to draw from. Whenever a face-up card is taken, it is immediately replaced with a card from the top of the deck so that there are always 5 options for card selection. Face-up cards are important determining factor of the decisions of a player either for choosing from the 3 types of actions or choosing the specific card to draw (in a turn of drawing card action, players can choose to pick from the face-up cards or draw from deck). Each possible permutation of the 5 face-up cards is considered a unique state (rather than the combination). This is in accordance with the way to model the drawing face-up cards action, which will be elaborated later in this section. As a result, all possible states of face-up cards are equivalent to the total permutations of the face-up cards. The state of face-up cards can be denoted as $\boldsymbol{Z_F} = \{F_1, F_2, F_3, F_4, F_5\}$. Each spot of the 5 face-up cards can be taken by one of the R colored cards or the wild card or no card (in case that the card deck is running out). Hence, F_i has $(R + 2)$ possible state, $i = 1, 2, \ldots, 5$. Therefore, the possible number of states of face-up cards $|\boldsymbol{Z_F}|$ can be calculated by (3).

$$|\boldsymbol{Z_F}| = (R + 2)^5 \tag{3}$$

d: HAND CARDS

From the perspective of a player, its hand cards contain essential information defining the state of the game. The total number of hand cards can be any integer from 0 to the largest number of cards in the deck. In addition, the order of obtaining the cards is irrelevant. Therefore, hand cards are recorded by a counter of cards in different colors, which can be denoted as $\boldsymbol{Z_H} = \{\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_\mathrm{R}, \mathbf{H}_\mathrm{wild}\}$, where H_i is the number of hand cards in Color i $(i = 1, 2, \ldots, R)$, the possible states of $H_i \in \{0, 1, 2, \ldots, N_C\}$, H_{wild} is the number of wild cards in hand cards, the possible states of $H_{wild} \in \{0, 1, 2, \ldots, N_w\}$. The number of possible combinations of hand cards $|\boldsymbol{Z_H}|$ can be calculated by (4).

$$|\boldsymbol{Z_H}| = (N_C + 1)^R \cdot (N_W + 1) \tag{4}$$

e: DESTINATION TICKETS

The destination tickets possessed by a player are important information when defining the game state, as succeeding or failing to fulfil a destination ticket leads to significant effect on the outcome of victory points (a plus or minus difference). As there are multiple chances for a player to draw tickets during a game, the number of destination tickets is dynamic. The state of the destination ticket of a player is defined by whether each destination ticket is possessed by the player, i.e., $\boldsymbol{Z_D} = \{\mathrm{DES}_1, \mathrm{DES}_2, \ldots, \mathrm{DES}_D\}$, where DES_i is a binary value determining the state of possession of Ticket i $(i = 1, 2, \ldots, D)$. Since the complete set of destination tickets can be a priori knowledge, the total number of states in terms of the destination tickets $|\boldsymbol{Z_D}|$ can be calculated by (5).

$$|\boldsymbol{Z_D}| = 2^D \tag{5}$$

f: POSSIBLE DESTINATIONS

When a player chooses to draw destination tickets, N_P destination tickets are dealt to the player for selection. The player must keep at least one ticket, up to N_P tickets. This temporal information is only shown when the drawing destination tickets action is in progress. To make the dealt tickets visible to a player, a vector of N_P values indicating the information of the dealt tickets is introduced. i.e., $\boldsymbol{Z_P} = \{P_1, P_2, \ldots, P_{N_P}\}$. P_i specifies the ith ticket delt with the possibilities of being any of the tickets in the deck or none (considering the possibility of the ticket deck is run out). The total number of states of the possible destinations $|\boldsymbol{Z_P}|$ can be calculated by (6).

$$|\boldsymbol{Z_P}| = (D + 1)^{N_P} \tag{6}$$

The complete observable state is the combination of the states specified in all these dimensions, $\boldsymbol{Z} = \{\boldsymbol{Z_B}, \boldsymbol{Z_C}, \boldsymbol{Z_F}, \boldsymbol{Z_H}, \boldsymbol{Z_D}, \boldsymbol{Z_P}\}$. The complete size of the observable state space $|\boldsymbol{Z}|$ for a player can be calculated by (7).

$$|\boldsymbol{Z}| = |\boldsymbol{Z_B}| \times |\boldsymbol{Z_C}| \times |\boldsymbol{Z_F}| \times |\boldsymbol{Z_H}| \times |\boldsymbol{Z_D}| \times |\boldsymbol{Z_P}| \tag{7}$$

Although aiming for accurate estimate, this is still an over-estimation due to the way that the game is modelled. Some states may never be reached such as some board states, hand card states and destination ticket states. However, it is certain that all possible observable states are taken into consideration.

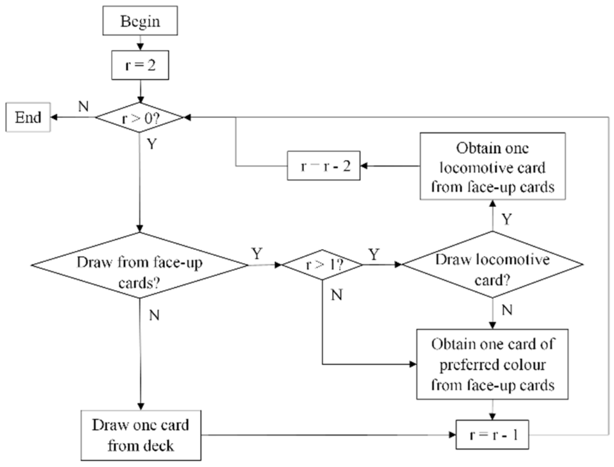


FIGURE 1. Decision process within a drawing train card turn. r is the number of remaining moves.

2) ACTION SPACE

Action space is composed of all the available actions of a player. Three types of actions from the original version of T2R are considered, i.e., drawing cards, drawing destination tickets, and connecting cities. Under each of these action dimensions, there are multiple possible action choices.

a: DRAWING TRAIN CARDS

At each turn of a player, the player can draw 2 train cards from deck or among face-up cards, except that only one card can be kept if a wild card is selected from face-up cards. The decision process for drawing card actions is shown in Fig. 1. Despite two opportunities of drawing cards being provided in each turn, we consider drawing each card an action. In other words, a player can perform 2 actions in a single turn if he/she chooses to draw cards. A player can choose to draw from deck or from face-up cards. In addition to the option of drawing from deck, drawing from face-up cards action can be modelled by specifying which of the 5 indexed spots to choose from (without considering the color of the cards), hence, another 5 possible options. Therefore, the total number of possible actions for a single drawing card action $|\boldsymbol{A_{tr}}|$ is 6.

Using the index of the 5 face-up cards as the representation of the drawing action is concise and straightforward. This modelling choice is made because the information regarding the sequence order of the face-up cards is kept in the state space (face-up cards dimension) and a player can also obtain the color information of a recently drawn card through the update of the hand card state.

b: DRAWING DESTINATION TICKETS

There are two consecutive steps within a single turn of a player performing a drawing destination ticket action.

In other words, the action is a 2-stage process. The first stage is to decide whether to draw tickets. If the decision is positive, the game state (possible destinations) needs to be updated as a number of tickets (N_P) will be dealt to the player. Then the second stage is to decide which ticket to keep considering the dealt tickets and current game status. Although both drawing train card actions and drawing ticket actions may involve 2 consecutive sub-actions, the action spaces of a drawing ticket action in the 2 stages are different since the player cannot see through the options in Stage 2 of drawing ticket action before going through Stage 1. The Stage 1 action is a single option ($|\boldsymbol{A}_d^1| = 1$) that can be made together with all the other possible actions of the other two types of actions. The Stage 2 action needs to be made according to a separate list of combinations of the indexes of the tickets (e.g., 0, 1, 2). For instance, if two tickets are dealt to the player ($N_P = 2$), the complete set of possible drawing ticket actions include 3 options ($|\boldsymbol{A}_d^2| = 3$), i.e., $\{(0), (1), (0, 1)\}$. The number of possible actions at Stage 2 can be calculated by (8).

$$|\boldsymbol{A}_d^2| = \sum_{i=1}^{N_P} \binom{N_P}{i} \tag{8}$$

The total number of actions related to drawing tickets $|\boldsymbol{A}_d| = 1 + |\boldsymbol{A}_d^2|$.

c: CONNECTING CITIES

The possible actions of connecting cities are dependent on the number of edges and the color of the edges. To connect a colored edge, cards of the same color are required and the number of cards of this color should meet the number of segments of this edge. Wild cards can be a replacement of any color as required. Therefore, in the case of connecting a colored edge, the action choice of connecting (cc) is determined by the number of wild cards used. To connect a no color edge (grey edge), cards of any same color which meet the number of segments of this edge are valid. There are also options to use wild cards as replacement instead for the chosen color. The number of action choices to connect an edge can be calculated by (9).

$$cc = \begin{cases} ns + 1 & \textit{if the edge is coloured} \\ R \cdot ns + 1 & \textit{if the edge has no colour} \end{cases} \tag{9}$$

By considering the possible connection actions for all edges, the total number of connecting actions ($|\boldsymbol{A}_c|$) can be calculated by (10).

$$|\boldsymbol{A}_c| = \sum_{i=1}^{m} ((ns_i + 1) \cdot \mathbf{1}\,(\textit{Edge i is coloured}) + (R \cdot ns_i + 1) \cdot \mathbf{1}\,(\textit{Edge i has no colour})) \tag{10}$$

The complete size of the action space $|\boldsymbol{A}|$ can be calculated by (11).

$$|\boldsymbol{A}| = |\boldsymbol{A}_{tr}| + |\boldsymbol{A}_d| + |\boldsymbol{A}_c| \tag{11}$$

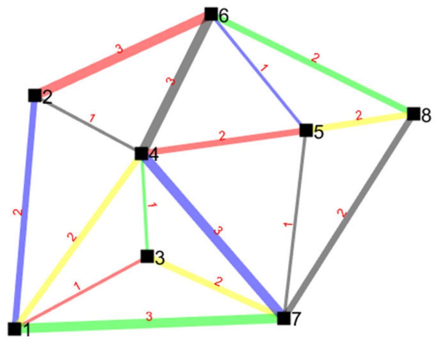


FIGURE 2. **Small map (length of each edge is labelled).**

B. CREATION OF A SCALED-DOWN VERSION OF TICKET TO RIDE

Due to the identified challenges in AI playing T2R, it is anticipated to be a difficult problem for RL (computationally costly) [17], [22]. Even the smallest commercial version of the game (Ticket to Ride: New York [94]) has 10^{49} estimated possible states and 235 possible actions according to (7) and (11). In comparison, chess has 10^{43} estimated possible states and 30 legal actions in a typical position [95]. Given a limited computational budget, one solution is to implement the proposed RL model in a scaled-down version of the game. As discussed in Section III-A, having a smaller map that consists of a reduced number of vertices, edges, cards, and train cars will result in fewer possible states and actions, namely, a smaller state-action space. The complexity of a sequential decision-making problem is associated with the size of state-action space [96]. For instance, in the context of DRL, more states result in extra work on learning a value network and an increased number of state-action pairs means a larger output dimension in the policy network. Hence, higher computational time is expected for solving problems with larger state-action space, and vice versa. Therefore, to fulfil the goal to demonstrate the capabilities of the proposed RL agents in playing T2R, a scaled-down map (referred to as Small Map) which is smaller than any commercial version of T2R is created (as shown in Fig. 2) based on game design principles used by a range of different versions (i.e., different maps or network layouts) of T2R (see details in Appendix A). All essential rules of T2R are strictly followed.

The configuration of Small Map is listed in Table 2. The map is an undirected, connected network made of 8 vertices and 16 edges. Each vertex is indexed with a distinct number (from 1 to 8). Out of the 16 edges, 12 are colored and 4 have no color (also known as grey edges). The 12 colored edges are evenly distributed in 4 colors (red, blue, yellow and green), i.e., 3 edges in each color. The number of segments in each color is exactly 6. Each edge is made of either 1, 2 or 3 segments. On average, each edge has 2 segments. Each vertex is connected to at least 3 other nodes. The degrees of the vertices range from 3 to 6. The most connected vertices are Vertex

TABLE 2. Configuration of small map.

Parameter	Value
Vertices (n)	8
Edges (m)	16
Number of Colors (R)	4
Number of Cards per Color (N_C)	6
Number of Wild Cards (N_W)	6
Number of Destination Tickets (D)	8
Number of Train Cars per Player (C)	10
Number of Colored Edges (m_c)	12
Number of Edges in Each Color	3
Number of Segments in Each Color	6
Number of Grey Edges (m_g)	4
Average Number of Segments per Edge ($\overline{ns_i}$)	2

TABLE 3. Destination tickets of small map.

No.	Destinations		Ticket	Shortest path	
	City 1	City 2	points	Segments	Alternatives
1	1	5	4	4	4
2	1	8	5	5	2
3	2	5	3	3	1
4	2	8	5	5	2
5	2	7	4	4	3
6	3	5	3	3	2
7	4	8	4	4	1
8	6	7	2	2	1

4 and Vertex 7, which are connected to 6 and 5 other nodes, respectively.

In addition to the graph features of the map, there are 6 train cards per color ($N_C = 6$), and 8 wild cards ($N_W = 8$). In total, there are 32 train cards in the deck. As listed in Table 3, 8 destination tickets ($D = 8$) are designed. Every vertex is mentioned by at least one ticket (and at most 3). The points on the tickets equal the number of segments of the shortest path connecting the destinations. Each ticket can be completed by connecting at least 2 edges. The average number of points of a ticket is 3.75. Each player starts the game with 10 train cars ($C = 10$).

Based on (7) and (11), the total number of observable possible states for a player in the simplified version of T2R is 1.7×10^{20} (the total number of possible states in a 2-player game is approximately 10^{27}). The number of possible actions is 78. Both the numbers of possible states and possible actions are significantly fewer than that of the smallest commercial version of T2R.

C. HEURISTIC AGENTS

Heuristic agents employ heuristic functions for action selection. Heuristic functions provide relational representation of states, actions and policies that are used to provide a natural abstraction of the domain. As the core of the action selection policy of a heuristic agent, a heuristic function can be derived from a preference policy, called heuristic policy π^H, which indicates that a preference of a certain action over others. Such a non-deterministic policy can be handcrafted or derived in several different manners [17], [38].

Six handcrafted heuristic AI agents are used in this paper. The first five are introduced in the T2R implementation developed by Schwarcz et al. [32]. Each of these heuristic agents make their actions based on a distinct cost function. The sixth heuristic agent is an ensemble of the other heuristic agents. These heuristic agents are described as follows (the details of the heuristic agent are specified in Appendix B):

1) CFBaseAI

CFBaseAI agent follows the if-then rules in the heuristic decision tree. When it comes to the important decision nodes, it will employ a specified cost function to evaluate the possible choices and choose the appropriate branch to perform the action. Firstly, it keeps updating the best path (a subset of the network) to complete its destination tickets according to the evaluation of the cost function. Then, if this best path can be located, it will choose the edge that costs least resources (train cards) according to the heuristic values assigned to the train cards. If there are not enough cards, it will get the cards needed for the given path. If the best path cannot be found, it will connect the edge with the best score when there are outstanding destination tickets, or it will draw more destination tickets.

2) CFRandomAI

CFRandomAI also selects the action during each step of the searching space following the heuristic decision tree but it makes random decisions at key decision nodes. After obtaining the best path with the same cost function as CFBaseAI, it makes random selection on the edge to connect among the candidate edges identified by the best path. Also, CFRandomAI draws train cards and ticket cards randomly if the decision tree has led it to these types of actions. Thus, it may heavily depend on luck and introduce more uncertainty of the playing.

3) CFActionEvalAI

CFActionEvalAI evaluates all valid actions given the temporal game state according to customized heuristic functions for each type of action. CFActionEvalAI then selects the best action according to the value evaluation by examining a wide range of possibilities. This may offset the deterministic features of such rule-based heuristics, but it requires a large amount of fine tuning for the parameters in the heuristic functions.

4) CFAdversarialAI

Like CFActionEvalAI, CFAdversarialAI also evaluates all the possible actions in terms of the current game state, but it will not perform the action with the highest value score all the time. Instead, it will first evaluate the threatened edge of the opponent and if it is claimable, the agent will claim the edge. Otherwise, it will perform the same as CFActionEvalAI.

5) CFCombinedAI

CFCombinedAI agent also takes the threatened edges into consideration. Instead of trying to directly claim the threatened edges, it assigns a higher weight to the threatened edges and evaluates the edges in combination with the action evaluation function applied by CFActionEvalAI.

6) CFGameTreeAI

CFGameTreeAI is an ensemble of the other heuristic agents. At each turn, the suggested actions of all the heuristic agents are generated. Like a voting system, CFGameTreeAI determines an action based on the majority principle.

D. RL AGENTS

Reinforcement learning (RL) is used in this paper to train an AI player to play T2R. Proximal Policy Optimization (PPO) [90] is a state-of-the-art general purpose RL algorithm used to train the RL agents. The PPO is implemented with a Multi-Layer Perceptron (MLP) policy as the state space and action space are discrete. As a part of the environment, the opponent player plays a pivotal role in the training process of the competing game. In this paper, only the 2-player setting is focused. Eight RL Training Schemes (RLTS) defined by the training partners (opponents) of the RL agent are introduced. The heuristic algorithms elaborated in Section III-C are used as the training partners as they have shown reasonable levels of skills and have different strategical emphases in their game play. Each of the 6 heuristic algorithms/agents is used as the training partner in an RLTS. For convenience, each of these RL agents is also referred to according to the corresponding training scheme, i.e., RLRandomAI, RLBaseAI, RLActionEvalAI, RLCombinedAI, RLAdversarialAI, and RLGameTreeAI. For instance, RLRandomAI refers to the RL agent which uses CFRandomAI as its training partner.

The 7th RLTS named *RLEnsembleAI* uses a combination of 5 heuristic agents as the training partner (all heuristics excluding CFGameTreeAI as CFGameTreeAI heuristic algorithm is an ensemble of the other 5 heuristics). Each of the 5 heuristics take turns to be the opponent of the RL agent for a fixed number of timesteps (T_e).

The 8th and final RLTS named *RLSelfplayAI,* is a co-evolutionary approach (or self-play), which does not involve any heuristic agents as the training partner [14], [31], [97]. Rather, it allows the RL agent to compete with a previous version of the RL agent itself during the training process. As illustrated in Fig. 3, the RL agent starts training from zero knowledge (Gen 0) with a random opponent. After training for several timesteps (T_s), the original RL agent has evolved to Gen 1 and the training opponent will be replaced by the last generation of RL agent (Gen 0). The process iterates until reaching the last generation. In other words, RLSelfplayAI keeps learning continuously but the model is capsulated every T_s timesteps and this model will perform as the training opponent in the next checkpoint. At any time point, the opponent of an RL agent is a player which plays the game according to the RL model obtained with at least T_s fewer timesteps.

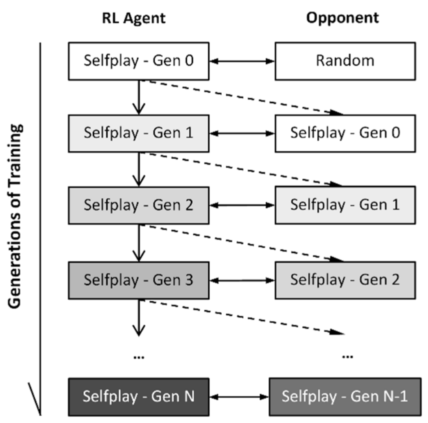


FIGURE 3. RLSelfplayAI training scheme.

IV. EXPERIMENTAL INVESTIGATION

Two sets of experiments are conducted to empirically demonstrate the capacity of Reinforcement Learning (RL) agents to learn good policies for Ticket to Ride (T2R) using the framework articulated above. The first experiment involves training different RL agents through different training schemes and testing the achieved RL agents by playing against a range of heuristic players. The second experiment further examines the capabilities of the RL agents beyond opponents with heuristic policies through a Round Robin tournament among all the obtained RL agents.

Two assumptions are taken in the experiments: Firstly, it is assumed that the simplified version of T2R (Small Map) is a good representation of the T2R games given that its creation follows the extracted design principles (similar graph attributes) of various versions of T2R; secondly, based on the limited observations of game data, it is assumed that the heuristic agents that were designed for commercial versions of T2R also possess a similar level of competence in playing the simplified version of T2R.

A. EXPERIMENTAL SETUP

The experimental setup is specified in this section including the setup with respect to the RL algorithm, design of reward system, and other parametric configurations.

1) RL IMPLEMENTATION

The RL is implemented through the online open-source RL implementations, Stable Baselines 3 [98], [99]. This Pytorch based package provides ample options for different RL algorithms and their customizations. A customized environment of T2R is created and synthesized with Stable Baselines 3. PPO is set to be the RL learner. Two fully connected layers of 64 nodes are used in both actor and critic networks. Learning rate is 3e-4. Discount factor 0.99. Clip range is 0.2.

TABLE 4. **Scoring system for edge building.**

Route (edge) length	Points/reward
1	1
2	2
3	4

The valid actions at each turn are only a small subset of the complete set of possible actions, with the list different at each turn. The valid state of each action is determined by the valid action function $L(p, a, s)$, with invalid actions masked, and therefore, unavailable for the player [100].

2) REWARD SYSTEM

The RL agents receive rewards from the environment every time after performing an action. The value of the reward is crucial for the learning process and has a great impact on the training outcome. The reward system applied in the experiment adopts reward shaping with a combination of immediate reward and delayed reward which basically follows the scoring system of T2R [101]. Two types of rewards are available to the learning player. Firstly, an immediate reward is received once it builds a route (edge) with the same value determined by the scoring system of T2R (as shown in Table 4). The second type of reward is from the completion of destination tickets. In the game, the ticket points are finalized at the end of the game. Completed tickets get positive points stated on the ticket and uncompleted tickets get the same negative points. A simple reward shaping is applied: At the moment a ticket is drawn, a negative immediate reward worth the points of the ticket is obtained; whenever the ticket is completed, a positive reward worth double the points of the ticket is gained. By doing so, the reward for a ticket is reshaped and applied at the two critical time points, i.e., at the ticket acquirement and the ticket completion. Other than the turns in which these two types of rewards are received, the reward of the turn will be 0.

3) EXPERIMENT PARAMETERS

For each of the RL agents, the total training timesteps are 1 million. Each timestep stands for an action of the RL agent. For the RLEnsembleAI, it is trained with alternating heuristic opponents. Each of them takes a turn of 10,000 timesteps ($T_e = 10{,}000$). For the RLSelfplayAI, every 1000 timesteps the opponent upgrades to the next generation ($T_s = 1000$).

Models obtained at various stages of the training process are used to test the performance of the RL agent. These models are saved every 10,000 timesteps. As a result, 100 models were kept at 100 checkpoints. In the testing phase, each of the 100 models was used for guiding agent's actions for 1000 timesteps. In other words, the results at each checkpoint are obtained from these 1000 timesteps. Moreover, 30 testing runs are conducted with each of the 100 models for statistical significance.

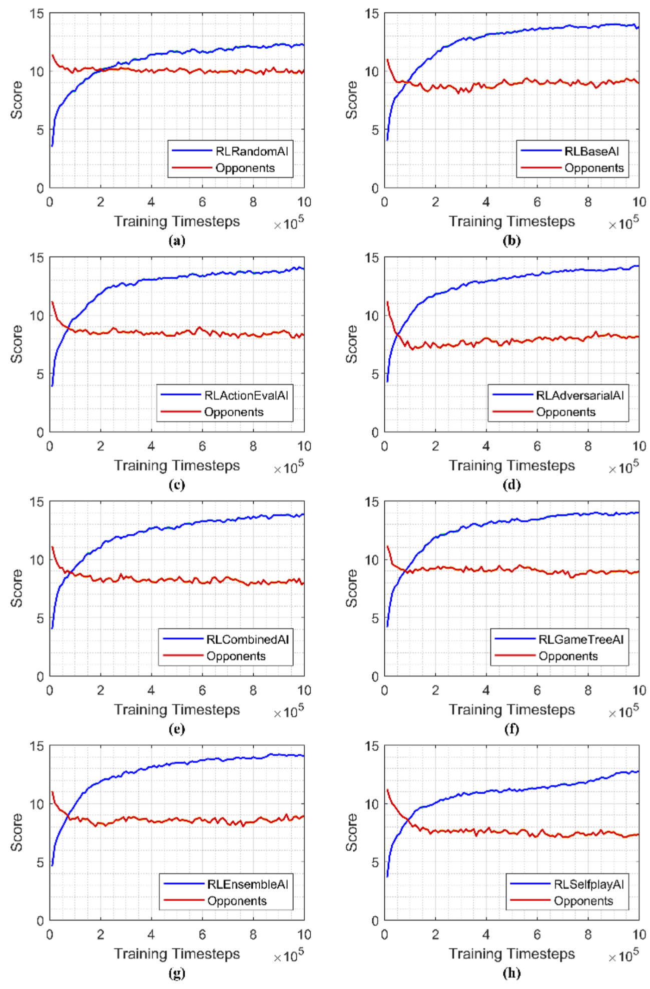


FIGURE 4. **Average score of each RL agent and the opponents across the training process.**

B. RESULTS

The results of the two sets of experiments are presented in this section, including the testing results obtained by the RL agents and the Round Robin tournament among the obtained RL agents.

1) PERFORMANCE OF RL AGENTS

The performance of each RL agent is evaluated based on the average performance versus the complete set of heuristic opponents. The average performance achieved by each RL agent is recorded across the training process.

As can be seen in Fig. 4, the average score per game of every RL agent keeps an increasing trend throughout the training process. With more training, the delta of the increase keeps decreasing. In contrast, starting at a winning position, the heuristic opponents' average score experiences a rapid decrease at the early stage of the training and maintains a significantly lower score than the RL agent for the rest of the training process. The average score of each RL agent surpasses that of the heuristic opponents within the first one fifth of the complete training process.

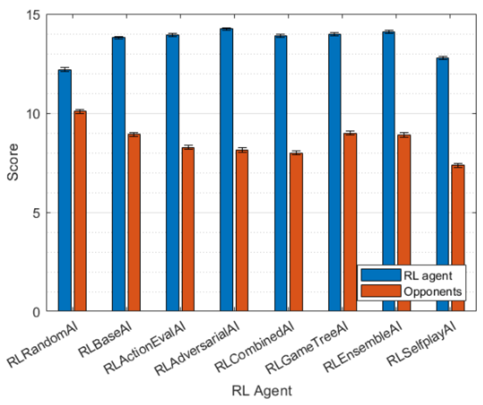


FIGURE 5. Average score of the RL agents and their opponents.

When the training is completed, the average score per game of each RL agent over the set of heuristic opponents is obtained as well as the performance of the corresponding opponents (as shown in Fig. 5). The average scores of the RL agents are at least 12.2 points, which is higher than the highest average score of any heuristic agent throughout the training. In fact, except for RLRandomAI (12.2) and RLSelfplayAI (12.8), the average scores of all the RL agents fall into a narrow range from 13.8 to 14.2. For every RL agent, after the training, its average score is significantly higher than the score of the heuristic opponents (approximately 50% higher in all but one case). Such an advantage over the opponents is larger than 4.9 points for all RL agents except for the RL agent trained with CFRandomAI as only a 2.1-point margin can be achieved versus its heuristic opponents.

As shown in Fig.4 and Fig. 6, at the beginning of the training, heuristic agents have a clear advantage over RL agents since RL agents perform almost randomly at this stage. The average win rate of each heuristic agent versus such random opponents is at least 0.75 and up to 0.91 (see Fig. 7). This demonstrates the validity of the assumption about the suitability of the heuristic agents in the scaled-down version of the game. With more training, the win rate of every RL agent shows a consistent increasing trend although the delta of the increase keeps decreasing. Asymptotic behavior is apparent in all cases by the end of training.

After the training, the final win rate achieved by each RL agent versus all heuristic agents is shown in Fig. 8. RLAdversarialAI (0.79) achieves the highest average win rate whereas RLRandomAI (0.57) achieves the lowest. Other than RLRandomAI, the win rate of all the RL agents ranges from 0.74 to 0.79. Although RLSelfplayAI tends to achieve lower scores than the other RL agents, its average win rate (0.75) is one of the highest among all eight RL agents.

The results are further analyzed for each matchup between an RL agent and a heuristic agent. As illustrated in Fig. 9, among all matchups, the average scores of the RL agents vary in a range from 10.6 to 14.7. RLRandomAI and RLSelfplayAI tend to achieve lower scores as their average scores versus the opponents range from 10.6 to 12.7 and from 12.3 to 13, respectively. RLRandomAI achieves the lowest average score (10.6) among all matchups when playing with CFAdversarialAI. It is also noticed that the average scores of RL agents tend to be lower when they compete with CFRandomAI and CFAdversarialAI. This indicates the

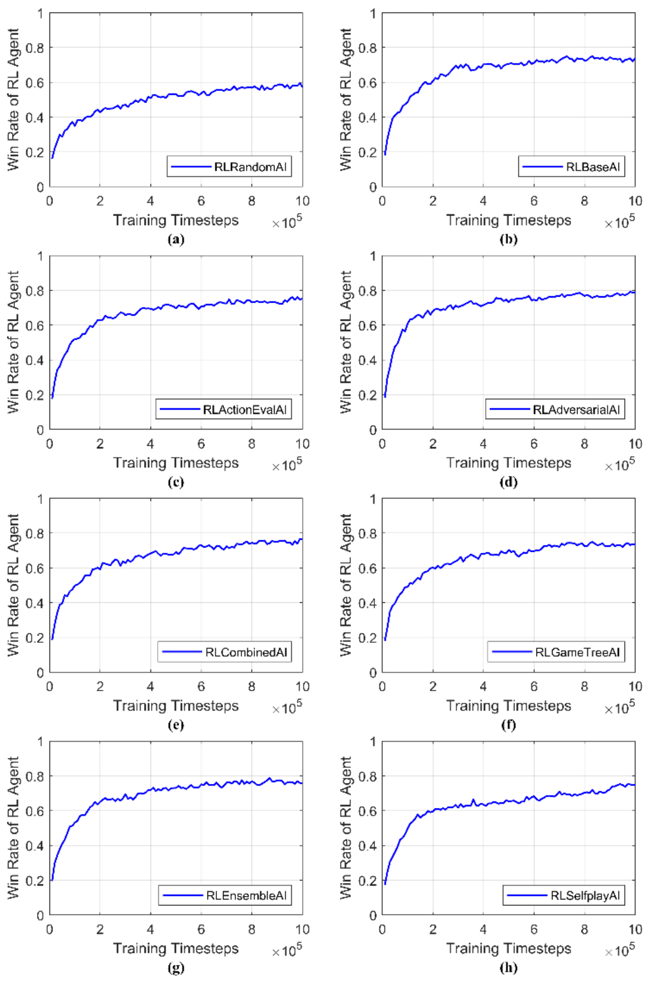


FIGURE 6. Average win rate of each RL agent across the training process.

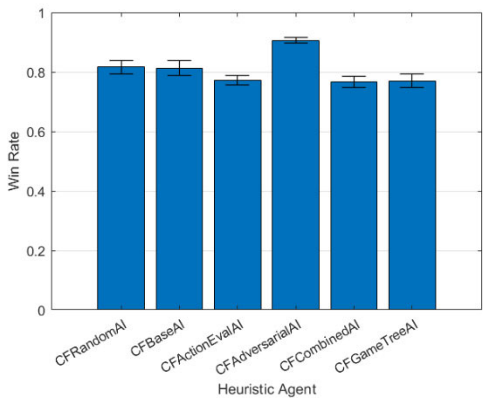


FIGURE 7. Average win rate of heuristic agents versus random players.

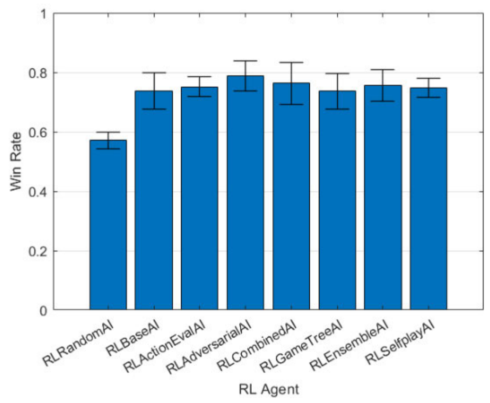


FIGURE 8. Win rate of each RL agent averaged across all the heuristic opponents.

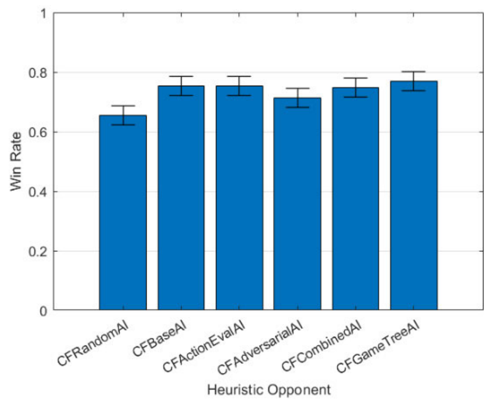


FIGURE 11. Average win rate (of the entire pool of RL agents) versus each heuristic opponent.

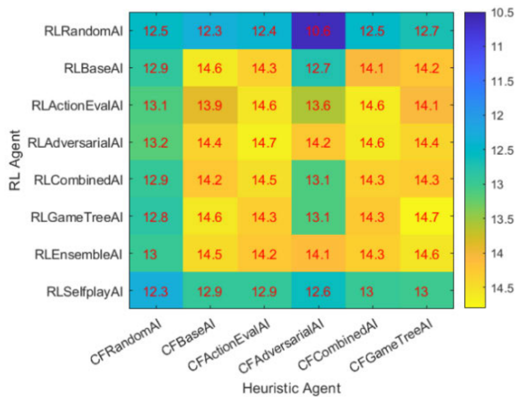


FIGURE 9. Average score achieved by RL agent versus different heuristic opponents.

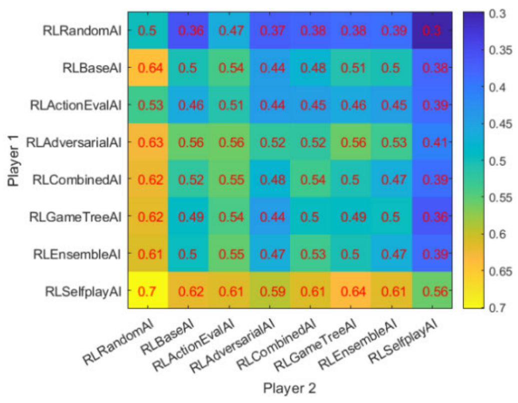


FIGURE 12. Average win rate (of Player 1) in a Round Robin between two RL players.

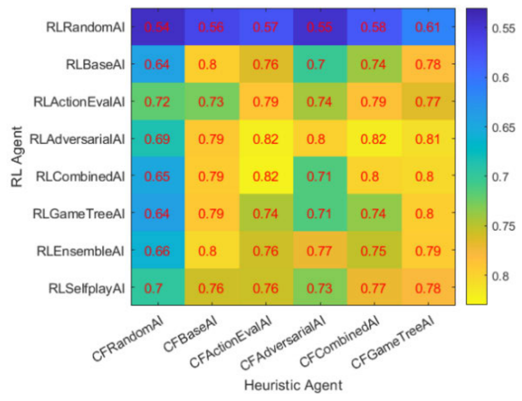


FIGURE 10. Average win rate achieved by RL agent versus different heuristic opponents.

higher difficulty for RL agents to achieve high score against CFRandomAI and CFAdversarialAI.

The win rates of all the matchups are shown in Fig. 10. The win rate achieved by RLRandomAI are significantly lower than any other RL agents in all scenarios with any heuristic opponent as the win rate ranges from 0.54 to 0.61. In contrast, although RLSelfplayAI achieves low average score (comparable with RLRandomAI), its win rate is comparable with most of the other RL agents, ranging from 0.7 to 0.78. The win rate of RL agent tends to be lower when the opponent is CFRandomAI or CFAdversarialAI, especially CFRandomAI.

As shown in Fig. 11, through differentiating the heuristic opponent, on average, the lowest average win rate can be expected when RL agents compete with CFRandomAI (0.65) compared to their average win rate achieved with the other heuristic opponents (ranges from 0.71 to 0.77).

2) ROUND ROBIN TOURNAMENT

A Round Robin tournament is conducted among all the RL agents. Each of the agents has the chance to start the game first (referred to as Player 1). The agent who starts next is referred to as Player 2. Each match between two players is conducted for 30,000 timesteps (which equates to between 2054 and 2222 games). The results shown are an average across all the games that happen within the specified timesteps.

As shown in Fig. 12, the win rate matrix is symmetric. In contrast to its relatively poor performance compared to the other RL agents in competition to their heuristic counterparts, RLSelfplayAI is the only RL agent which achieves advantages over all the other RL agents in terms of win rate as its win rate versus any other RL agent is at least 0.59 and up to 0.7. RLRandomAI performs poorly versus most the other RL agents as it only achieves at most a 0.39 win rate over any other RL agent except for RLActionEvalAI (0.47). In other matchups, the performance of both players is close as the average win rate ranges from 0.44 to 0.56.

The low win rate achieved by RLRandomAI is consistent with the significantly worse performance versus the heuristic agents. However, while RLSelfplayAI achieves the best performance against the other RL agents, that is not the case against the heuristic opponents. It is believed that this RL agent possesses some uniqueness compared to the RL agents and the heuristic agents. Since RLSelfplayAI is not trained with any heuristic agent but with a previous version of itself, its training experience does not overfit to the constrained paradigms resulting from the hard-coded rules of the heuristic opponents. Hence, RLSelfplayAI is more versatile and can deal with various opponents, including both different heuristic agents and RL agents.

V. CONCLUSION

This paper demonstrated the capabilities of AI agents trained with general-purpose RL algorithm (PPO) in playing a complex imperfect information board game with delayed rewards – Ticket to Ride (T2R) with no domain knowledge beyond game rules, game states and scores. Inductive bias is only introduced when necessary, such as a simple reward shaping and the intuitive ways of encoding state-action space, which are reflections of the game rules. Eight RL agents were obtained through different training schemes, which were defined by the opponent involved in the training under the 2-player setting. It was demonstrated that all these RL agents achieved advantage over a set of heuristic opponents including the RL agent trained with a self-play strategy (RLSelfplayAI). Furthermore, RLSelfplayAI was demonstrated to be the best RL agent when competing with any of the RL agents. It was the only RL agent that achieved advantages over all the other RL agents. Moreover, RLSelfplayAI has shown better versatility in competing with different opponents including both heuristics and RL agents. This suggests that without involving any heuristics, such as opponents with hard-coded heuristic rules for playing T2R, the general-purpose RL algorithm (PPO) can still achieve high performance and versatility through self-play. It is also recognized that there is still room to improve the training scheme of RLSelfplayAI as evidenced by its relative performance against heuristic agents.

Future work is planned. Firstly, the success of RL in this simplified version of T2R presents an opportunity to take the same method to a larger implementation of the game. The next step will be the demonstration of the capabilities of RL agents in the commercial versions (larger maps) of T2R. Secondly, multiplayer games are more general situations and will unfold more possibilities than 2-player games. Future work on the multiplayer settings instead of the current 2-player setting will be conducted by further improving the RL algorithms and training schemes. Thirdly, previous states were not taken into account in training the RL policy in this implementation. However, past states are suspected to be important in inferring belief states in Partially Observable Markov Decision Processes (POMDP) [102]. Therefore, using historic data sequence through Recurrent Neural Networks (RNNs) may advance the current RL algorithm in tackling T2R.

APPENDIX

A. GAME DESIGN PRINCIPLES

As one of the most successful board games in the world, the elegant design of T2R has been celebrated by players around the world since the success of the original version in 2004. To accommodate the enthusiasm, a variety of versions of the game have been released in the last two decades. In the hope of understanding the design choices, this section investigates different versions of T2R from several essential aspects of the game design in terms of the numeric and graphic parameters.

1) NUMERIC PARAMETERS OF VARIOUS TICKET TO RIDE VERSIONS

The numerical parameters of a considerable range of different Ticket to Ride versions are summarized in Table 5. Further analysis of these numerical configurations is given below.

a: VERTICES

Among the parameters, the number of vertices/cities/regions (n) is an independent variable to the other parameters. The number of cities determines the complexity/difficulty of the game/task to a significant extent. Among the investigated versions, it varies from 14 to 47.

b: EDGES

All vertices on a T2R map are connected through edges. The choice of connecting cities with edges is reflected by the distribution of degrees (refer to the next section for further analysis on degrees). In all analyzed versions, the number of edges is approximately 2 times the number of vertices ($2n$). Moreover, the sum of degrees of all vertices is 2 times the number of edges. As a result, the average value of the degree of a city is 4, i.e., on average, each city has 4 edges or is connected to 4 other cities.

c: NUMBER OF DESTINATION TICKETS

The number of destination tickets approximates n in most investigated versions. The set of tickets in each version tends to nominate each city in the map as a destination at least once.

d: NUMBER OF COLORS

The colors on edges are in accordance with the colors on train cards. The number of colors is either 6 or 8. The choice is related to the size of the map (the number of vertices).

TABLE 5. Configuration summary of different versions of ticket to ride.

	Demo – North America	Demo – Europe	New York	London	Amsterdam	First Journey	Poland	Original	Germany	Legendary Asia	Italy	Team Asia	Europe
Vertices	14	15	15	17	17	19	35	36	36	39	42	46	47
Edges[a]	27	29	30	35	32	39	68	73	80	70	84	90	90
Edges[b]	41	42	41	44	41	64	102	94	105	100	126	130	101
Double Edges[c]	14	13	11	9	9	25	21	21	21	30	40	32	11
Number of Colors	6	6	6	6	6	6	8	8	8	8	8	8	8
Number of cards per color	6	6	6	6	6	-	12	12	12	12	-	12	12
Number of wild cards	8	8	8	8	8	-	14	14	14	14	-	14	14
Number of Destination Tickets	18	18	18	20	24	32	35	30	89	36	56	60	46
Number of Train Cars per Player	15	15	15	17	16	20	35	45	45	45	45	45	45

a Counting double/triple/quadruple edges once.
b Counting double edges twice, triple edges 3 times, etc.
c Including triple and quadruple edges.

TABLE 6. Map attribute analysis of different versions of ticket to ride.

	Demo – North America	Demo – Europe	New York	London	Amsterdam	First Journey	Poland	Original	Germany	Legendary Asia	Italy	Team Asia	Europe
Mean Degree	4	4	4	4	4	4	4	4	4	4	4	4	4
Min Degree	2	3	3	3	2	3	1	2	2	2	1	1	1
Number of Min Degree	2	9	6	4	1	8	7	4	1	7	4	1	1
Max Degree	6	7	6	6	5	6	6	7	7	6	6	7	7
Number of Max Degree	3	2	1	2	2	3	6	1	2	3	9	1	1
Min Edge Length	1	1	1	1	1	1	1	1	1	1	1	1	1
Max Edge Length	4	4	4	4	4	4	5	6	7	6	7	7	8
Number of Edges in Each Color	6	7	5	5	4	11	11	7	8	9	13	13	8
Number of Segments in Each Color	12	13	11	11	9	22	25	27	28	25	34	29	22
Number of Grey Edges	3	2	8	14	13	0	13	43	32	27	27	24	36

In specific, versions with 35 or more vertices have 8 colors, whereas the smaller maps have 6 colors.

2) GRAPHIC ATTRIBUTES

In addition to the numeric parameters regarding the general settings, further analysis is conducted on a variety of graphic attributes that further define the dynamics of the game (as shown in Table 6). A large map (Europe version) and a small map (New York version) are taken as examples in some in-depth analysis.

a: DEGREE DISTRIBUTION

The level of degree of a vertex in a map is a result of the design choices made regarding spatial connections between cities and edges. Some statistics of the distribution of degrees are summarized in Table 6 including mean degree, minimum degree, maximum degree, number of vertices with the minimum degree, number of vertices with the maximum degree. The degrees are calculated by only counting double/triple/quadruple edges once. Although the distribution of degrees varies in different versions of T2R, their mean degrees are all 4. In other words, the higher complexity of a version with a larger map is not introduced by increasing the degrees of vertices. The number of vertices is still the dominant factor.

b: EDGE COLOR DISTRIBUTION

The number of edges in each color tends to be the same in each version with a few biases. The number of edges in each color differs across different versions. It is a function of the total number of edges and the number of edge colors.

Similarly, the number of segments in each color (including all edges in the color) tends to be around the same value in each version. This design principle is more consistently adopted than that around the number of edges in each color since the bias tends to be smaller.

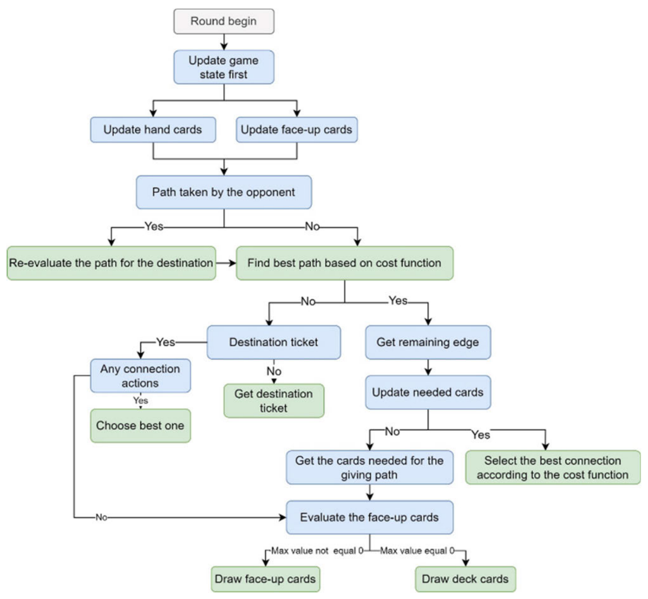


FIGURE 13. Visualization of the heuristics used for CFBaseAI.

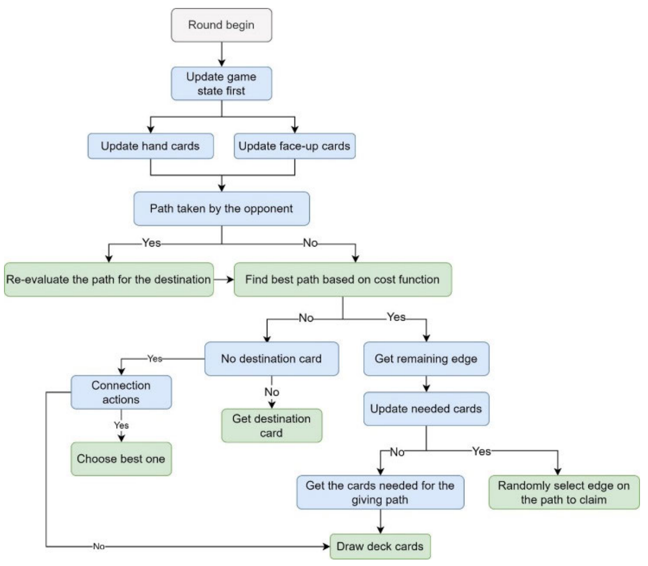


FIGURE 14. Visualization of the heuristics used for CFRandomAI.

For edges without color (grey edges), no clear pattern is observed in terms of number of grey edges across different versions.

c: TICKET POINTS VS. SHORTEST PATH

Based on analysis on Europe version and New York version, for most destination tickets (Europe: 35/46, New York: 12/18), there is only one shortest path in terms of segments. This is achieved through the highly asymmetric structure of the train networks.

Furthermore, based on analyses on Europe version and New York version, the number of points of a ticket equals the segment length of the path (connecting the 2 destinations of the ticket) with the fewest segments, or the least number of train cars to complete the ticket/contract (only with 3 exceptions with minor bias in Europe version).

d: DOUBLE EDGES VS POPULARITY

The number of double edges varies from version to version. There is a weak correlation between the choice of a double edge with the number of times visited by the shortest path of a destination ticket (0.37 for Europe version and 0.20 for New York version). In other words, if an edge is visited by more shortest paths, it is more likely that it is a double edge.

B. DESCRIPTION OF THE HEURISTIC AGENTS

1) CFBaseAI

CFBaseAI has a set of rules and a cost function to make decisions in each round. CFBaseAI plays the game by following the rules described in Fig. 13. It first attempts to examine whether the desired path for completing destination cards has been taken by the opponent, if yes, the agent will re-evaluate the path for the destination cards, otherwise it will find the best path according to the value of cost function, as described in Algorithm 1.

If the agent can find the path based on the cost function, then it calculates the available edges and chooses the edge with the least cost value to claim (as described in Algorithm 2). But if there are not enough cards, it will get the cards needed for the given path. By doing that, the agent will evaluate the value of face-up cards first, if the value of face-up card is larger than 0, it will select face-up cards, otherwise the agent will pick up cards from the deck.

If the agent cannot find the path based on the cost function, it will pick the destination ticket if there is not enough destination ticket in hand. Otherwise, it will claim a connection with the best score if there is any available destination ticket.

Algorithm 1 Cost Function for Evaluating Individual Paths

```
Input: the path to calculate cost, the list of all path, a dictionary
containing the costs of all edges, hand cards, face-up cards,
Edge_Color_Exp
Output: cost of the path
possible_cards = [hand_cards face_up_cards]
for each card in possible_cards
        If card in cards needed
                card_needed −= card
                cards_needed_num −= 1;
                useful_card_num += 1;
end for each
for each card in card_needed do
        If needed _ cards in grey color
                Cost += card_needed_num;
        else
                Cost += card_needed_num ^Edge_Color_Exp;
end for each
```

The way to evaluate the face-up cards is described as follows: the agent will go through each card in the face-up cards by first examine whether the card is wild card or not, if it is and the remaining action is two (no card has been drawn), the value will plus wild card value, but if it is and

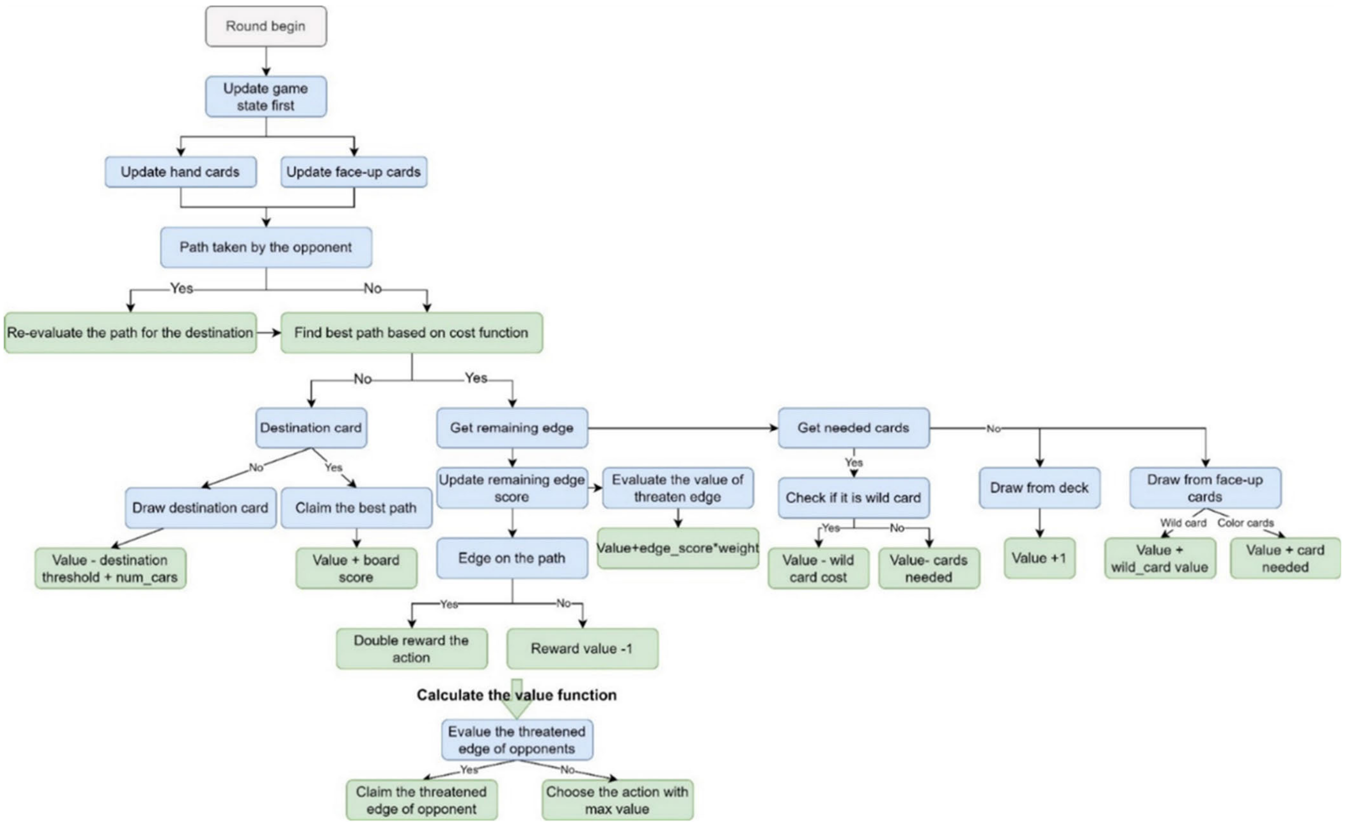


FIGURE 15. **Visualization of the heuristics used for CFAdversarialAI. The heuristic for CFActionEvalAI is the same except the step to evaluate the threated edge of opponents is removed.**

remaining action is one (one colored card is drawn or one card is drawn from the deck), the value will decrease one. If the card is not a wild card, the value will plus the card value if it is the needed card. Thus, the face-up cards will be evaluated and if the value is 0, the agent will select cards from the deck.

2) CFRandomAI

The design of CFRandomAI is based on the CFBaseAI, as depicted in Fig. 14. The workflows of those two agents are similar except that after evaluating the best path based on the cost function, randomness is introduced to finalize the actions. To be more specific, CFRandomAI will choose random actions when connecting an edge for best path and drawing train cards. Besides, when there are not enough cards to claim the edge, the agent will randomly select cards from the deck instead of selecting cards from the face-up cards.

3) CFActionEvalAI

CFActionEvalAI uses a set of cost functions to evaluate each action in each turn and select the best action. The novelty lies with two aspects: first is to evaluate each action based on the path and cost function. The agent will initially repeat the same process of CFBaseAI, updating the game states by finding best path, available destination cards and train cards. Then, the agent will get all the available actions based on the game state and calculate the value score for each possible action. As shown in Algorithm 3, the CFActionEvalAI agent has different heuristic mechanisms to calculate value scores for different actions given the temporal state. Fig. 15 presents the flowchart/decision tree of CFActionEvalAI, which is almost identical with CFBaseAI. The difference is that the CFActionEvalAI calculates the value score for each branch, then synthesize the results for the action space. Thus, the value score will be calculated for each action, and the agent will select the best action with the highest score to implement.

4) CFAdversarialAI

CFAdversarialAI plays almost the same way as the CFActionEvalAI, but it will predict the threatened edge of the opponent (steal edge) first before performing the original action with the best value. The agent will perform the value score calculation as the CFActionEvalAI, but it will evaluate the possible steal edges, and find an action to claim it. If the action is achievable, the agent will claim the threatened edge of the opponent first instead of claiming their best choice, as shown in Fig. 15.

5) CFCombinedAI

CFCombinedAI performs similarly to CFAdversarialAI, except that rather than directly claiming the threatened edge of the opponent, CFCombinedAI evaluates the value of the threatened edge with Algorithm 4 and combines this

Algorithm 2 Select Edge on the Path to Claim

```
Input: the path, the costs of all edges, Wild_card_cost,
Edge_Score_Multiplier, Edge_Score
Output: best action
cards_needed = get cards needed to connect the path
for each edge in path
    all_connection_actions = get all possible connects;
for each action in all_connection_actions do
    for card in action.cards (combinations of cards required by the connection actions):
        if card == wild card
            Cost += Wild_card_cost;
        else
            Cost += cards_needed[card];
        end
        Cost −= Edge_Score_Multiplier * Edge_Score[edge_cost]
    end for each
end for each
```

Algorithm 3 Evaluate Action Based on Path and Cost Function

```
Input: action to be evaluated, path, threat_action_weight,
Wild_card_value, Edge_Score, remaining edge, remaining_edge_score,
wild_card_cost, num_cars
Output: the value of the action
cards_needed = get cards needed to connect the path
if edge in threatened edges
    value += threatened_edges_score * threat_action_weight;
if path is not none :
    if edge in remaining edge do
        value += Wild_card_value + Edge_Score \ path score – remaining_edge_score:
    else edge not in remaining edge
        value += −1
    for each card in cards required by action
        if card is wildcard
            value −= wild_card_cost
        else
            value −= cards_needed
    end for each
if path is none:
    value += Edge_Score
if draw destination card
value −= destination_threshold + num_cars
if draw deck cards:
    value += wild_card_value
if draw face up cards:
    if draw wild_card:
        value += wild_card_value
    else
        value += cards_needed
```

score with the evaluation of the edge made by Algorithm 3. In Algorithm 4, first, the double edges are removed from the threatened edges because the double edge cannot be blocked instantaneously. Then, it will evaluate each edge in the edge groups by adding a score based on the player's remaining cards, adding score based on all the edge's cost and adding score based on the length of the edge group. If the score is less than 0, the score will be 0. As shown in Algorithm 3, if the action aims to claim the threatened edge, the value will plus threatened edge score * threat action weight, which indicates that the threatened edge will be given high priority. However, the threat action weight should be appropriately defined.

Algorithm 4 Evaluate Threatened Edge

```
Input: threatened edge to be evaluated, T_multi_edge_penalty,
T_Remaining_Cars, T_edge_weight, T_edge_group_length
Output: the value of the action
if threatened edge in double edges
    remove double edges;
penalty = (len(threatened_edges) − 1) * T_multi_edge_penalty:
for each temp_edge_group in threatened_edges:
    score = 0 – penalty
    score += T_remainig_cards * (num_total_cars – num_cars);
    for each edge_group in temp_edge_group:
        for each edge in edge_group:
            score += edge cost * T_edge_weight;
        end for each
        score += len(edge_group)^(T_edge_group_length);
        score = 0;
    end for each
end for each
```

6) CFGameTreeAI

CFGameTreeAI is an ensemble approach which gives equal weight to the decisions of the previous five heuristic agents. The action with the highest votes will be performed.

REFERENCES

[1] R. Rasmussen, "A game theory approach to high-level strategic planning in first person shooters," in *Proc. 5th Australas. Conf. Interact. Entertainment*, Brisbane, QLD, Australia, Dec. 2008, pp. 1–4.

[2] M. Sousa and E. Bernardo, "Back in the game: Modern board games," in *Proc. VIDEOJOGOS*, Aveiro, Portugal, 2019, pp. 72–85.

[3] QE. *Board Game Geek*. Accessed: Mar. 23, 2023. [Online]. Available: https://boardgamegeek.com/boardgame/266830/qe

[4] *COIN Series*. Accessed: Mar. 23, 2023. [Online]. Available: https://www.gmtgames.com/c-36.aspx?searchEngineName=coin-series#[PageNumber(0)|PageSize(50)|PageSort(Name)|DisplayType(Grid)

[5] *Board Game Geek, Cuba Libre*. Accessed: Mar. 23, 2023. [Online]. Available: https://boardgamegeek.com/boardgame/111799/cuba-libre

[6] X. Qi. *Board Game Geek*. Accessed: Mar. 23, 2023. [Online]. Available: https://boardgamegeek.com/boardgame/2393/xiangqi

[7] G. Moy and S. Shekh, "The application of AlphaZero to wargaming," in *Proc. AJCAI*, Adelaide, SA, Australia, 2019, pp. 3–14.

[8] J. Goodman, S. Risi, and S. Lucas, "AI and wargaming," 2020, *arXiv:2009.08922*.

[9] M. L. Ginsberg, "GIB: Imperfect information in a computationally challenging game," *Artif. Intell. Res.*, vol. 14, pp. 303–358, Jun. 2001.

[10] I. Szita, G. Chaslot, and P. Spronck, "Monte-Carlo tree search in settlers of Catan," in *Proc. ACG*, Pamplona, Spain, 2009, pp. 21–32.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[12] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-up limit hold'em poker is solved," *Science*, vol. 347, no. 6218, pp. 145–149, Jan. 2015, doi: 10.1126/science.1259433.

[13] M. Campbell, A. J. Hoane Jr., and F.-H. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, nos. 1–2, pp. 57–83, Jan. 2002, doi: 10.1016/S0004-3702(01)00129-1.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[15] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, ''The grand challenge of computer Go: Monte Carlo tree search and extensions,'' *Commun. ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012, doi: 10.1145/2093548.2093574.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, ''Mastering the game of go without human knowledge,'' *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.

[17] F. D. M. Silva, S. Lee, J. Togelius, and A. Nealen, ''AI as evaluator: Search driven playtesting of modern board games,'' in *Proc. Workshops AAAI*, San Francisco, CA, USA, 2017, pp. 959–966.

[18] C. Huchler, ''An MCTS agent for ticket to ride,'' M.S. thesis, Fac. Humanit. Sci., Maastricht Univ., Maastricht, The Netherlands, 2015.

[19] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, ''RUDDER: Return decomposition for delayed rewards,'' in *Proc. NeurIPS*, Vancouver, BC, Canada, 2019, pp. 13566–13577.

[20] C. Berner et al., ''Dota 2 with large scale deep reinforcement learning,'' 2019, *arXiv:1912.06680*.

[21] B. Han, Z. Ren, Z. Wu, Y. Zhou, and J. Peng, ''Off-policy reinforcement learning with delayed rewards,'' in *Proc. PMLR*, Baltimore, MD, USA, 2022, pp. 8280–8303.

[22] L. Strömberg and V. Lind, ''Board game AI using reinforcement learning,'' B.S. thesis, Sch. Sci. Technol., Örebro Univ., Örebro, Sweden, 2022.

[23] A. Goldwaser and M. Thielscher, ''Deep reinforcement learning for general game playing,'' in *Proc. AAAI*, New York, NY, USA, 2020, pp. 1701–1708.

[24] M. Schmid, M. Moravcik, N. Burch, R. Kadlec, J. Davidson, K. Waugh, N. Bard, F. Timbers, M. Lanctot, Z. Holland, E. Davoodi, A. Christianson, and M. Bowling, ''Player of games,'' 2021, *arXiv:2112.03178*.

[25] N. Brown and T. Sandholm, ''Superhuman AI for heads-up no-limit poker: Libratus beats top professionals,'' *Science*, vol. 359, no. 6374, pp. 418–424, Jan. 2018, doi: 10.1126/science.aao1733.

[26] N. Brown and T. Sandholm, ''Superhuman AI for multiplayer poker,'' *Science*, vol. 365, no. 6456, pp. 885–890, Aug. 2019, doi: 10.1126/science.aay2400.

[27] K. He, H. Wu, Z. Wang, and H. Li, ''Finding Nash equilibrium for imperfect information games via fictitious play based on local regret minimization,'' *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 6152–6167, Feb. 2022, doi: 10.1002/int.22837.

[28] N. Brown, A. Lerer, S. Gross, and T. Sandholm, ''Deep counterfactual regret minimization,'' in *Proc. PMLR*, Long Beach, CA, USA, 2019, pp. 793–802.

[29] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, ''Regret minimization in games with incomplete information,'' in *Proc. NIPS*, Vancouver, BC, Canada, 2007, pp. 1729–1736.

[30] M. Genesereth and M. Thielscher, ''General game playing,'' *Synth. Lect. Artif. Intell. Mach. Learn.*, vol. 8, no. 2, pp. 1–229, 2014.

[31] J. Heinrich and D. Silver, ''Deep reinforcement learning from self-play in imperfect-information games,'' 2016, *arXiv:1603.01121*.

[32] S. Schwarcz, J. Twigg, and D. Zeng. *Ticket to Ride Library*. Accessed: Feb. 21, 2023. [Online]. Available: https://github.com/willdzeng/ticket_to_ride

[33] Eurogame. *Board Game Geek*. Accessed: Feb. 21, 2023. [Online]. Available: https://boardgamegeek.com/wiki/page/Eurogame

[34] Ticket to Ride. *Board Game Geek*. Accessed: Feb. 21, 2023. [Online]. Available: https://boardgamegeek.com/boardgame/9209/ticket-ride

[35] R. S. Thomas, ''Real-time decision making for adversarial environments using a plan-based heuristic,'' Ph.D. dissertation, McCormick School Eng., Northwest. Univ., Evanston, IL, USA, 2003.

[36] K. Xenou, G. Chalkiadakis, and S. Afantenos, ''Deep reinforcement learning in strategic board game environments,'' in *Proc. EURAMAS*, Bergen, Norway, 2018, pp. 233–248.

[37] M. Pfeiffer, ''Reinforcement learning of strategies for Settlers of Catan,'' in *Proc. CGAIDE*, Wolverhampton, U.K., 2004, pp. 384–388.

[38] S. D. Palma and P. L. Lanzi, ''Traditional wisdom and Monte Carlo tree search face-to-face in the card game Scopone,'' *IEEE Trans. Games*, vol. 10, no. 3, pp. 317–332, Sep. 2018, doi: 10.1109/TG.2018.2834618.

[39] A. Uriarte and S. Ontañón, ''Single believe state generation for partially observable real-time strategy games,'' in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, New York, NY, USA, Aug. 2017, pp. 296–303.

[40] M. S. Campbell and T. A. Marsland, ''A comparison of minimax tree search algorithms,'' *Artif. Intell.*, vol. 20, no. 4, pp. 347–367, Jul. 1983, doi: 10.1016/0004-3702(83)90001-2.

[41] G. M. Adelson-Velsky, V. L. Arlazarov, and M. V. Donskoy, ''Two-person games with complete information and the search of positions,'' in *Algorithms for Games*, 1st ed. New York, NY, USA: Springer-Verlag, 2012, ch. 1, pp. 1–32.

[42] G. Tesauro and G. R. Galperin, ''On-line policy improvement using Monte-Carlo search,'' in *Proc. NeurIPS*, Cambridge, MA, USA, 1996, pp. 1068–1074.

[43] M. H. M. Winands and Y. Björnsson, ''Evaluation function based Monte-Carlo LOA,'' in *Proc. ACG*, Pamplona, Spain, 2010, pp. 33–44.

[44] R. Lorentz, ''Using evaluation functions in Monte-Carlo tree search,'' *Theor. Comput. Sci.*, vol. 644, pp. 106–113, Sep. 2016, doi: 10.1016/j.tcs.2016.06.026.

[45] A. Junghanns, ''Are there practical alternatives to alpha-beta?'' *ICGA J.*, vol. 21, no. 1, pp. 14–32, Mar. 1998, doi: 10.3233/ICG-1998-21103.

[46] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. V. D. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, ''Progressive strategies for Monte-Carlo tree search,'' *New Math. Natural Comput.*, vol. 4, no. 3, pp. 343–357, Nov. 2008, doi: 10.1142/S1793005708001094.

[47] B. Brügmann, ''Monte Carlo Go,'' Phys. Dept., Syracuse Univ., Syracuse, NY, USA, Tech. Rep., Oct. 1993. [Online]. Available: http://www.ideanest.com/vegos/MonteCarloGo.pdf

[48] R. Coulom, ''Efficient selectivity and backup operators in Monte-Carlo tree search,'' in *Proc. CG*, Turin, Italy, 2007, pp. 72–83.

[49] L. Kocsis and C. Szepesvári, ''Bandit based monte-carlo planning,'' in *Proc. ECML*, Berlin, Germany, 2006, pp. 282–293.

[50] D. Silver and J. Veness, ''Monte-Carlo planning in large POMDPs,'' in *Proc. NIPS*, Vancouver, BC, Canada, 2010, pp. 2164–2172.

[51] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, ''A general reinforcement learning algorithm that masters chess, Shogi, and Go through self-play,'' *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018, doi: 10.1126/science.aar6404.

[52] R. Coulom, ''Computing 'Elo ratings' of move patterns in the game of GO1,'' *ICGA J.*, vol. 30, no. 4, pp. 198–208, Dec. 2007, doi: 10.3233/ICG-2007-30403.

[53] S. Gelly and D. Silver, ''Monte-Carlo tree search and rapid action value estimation in computer Go,'' *Artif. Intell.*, vol. 175, no. 11, pp. 1856–1875, Jul. 2011, doi: 10.1016/j.artint.2011.03.007.

[54] M. Johanson, N. Bard, N. Burch, and M. Bowling, ''Finding optimal abstract strategies in extensive-form games,'' in *Proc. AAAI*, Toronto, ON, Canada, 2012, pp. 1371–1379.

[55] P. I. Cowling, C. D. Ward, and E. J. Powley, ''Ensemble determinization in Monte Carlo tree search for the imperfect information card game magic: The gathering,'' *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 4, pp. 241–257, Dec. 2012, doi: 10.1109/TCIAIG.2012.2204883.

[56] I. Frank and D. Basin, ''Search in games with incomplete information: A case study using bridge card play,'' *Artif. Intell.*, vol. 100, nos. 1–2, pp. 87–123, Apr. 1998, doi: 10.1016/S0004-3702(97)00082-9.

[57] P. I. Cowling, E. J. Powley, and D. Whitehouse, ''Information set Monte Carlo tree search,'' *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 120–143, Jun. 2012, doi: 10.1109/TCIAIG.2012.2200894.

[58] J. Long, N. R. Sturtevant, M. Buro, and T. Furtak, ''Understanding the success of perfect information Monte Carlo sampling in game tree search,'' in *Proc. AAAI*, Atlanta, GA, USA, 2010, pp. 134–140.

[59] H. Ihara, S. Imai, S. Oyama, and M. Kurihara, ''Implementation and evaluation of information set Monte Carlo tree search for Pokémon,'' in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Miyazaki, Japan, Oct. 2018, pp. 2182–2187.

[60] M. Buro, J. R. Long, T. Furtak, and N. Sturtevant, ''Improving state evaluation, inference, and search in trick-based card games,'' in *Proc. IJCAI*, Pasadena, CA, USA, 2009, pp. 1407–1413.

[61] J. Wang, T. Zhu, H. Li, C. Hsueh, and I. Wu, ''Belief-state Monte-Carlo tree search for phantom games,'' in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Aug. 2015, pp. 267–274.

[62] D. Tolpin and S. E. Shimon, ''MCTS based on simple regret,'' in *Proc. AAAI*, Toronto, ON, Canada, 2012, pp. 570–576.

[63] F. Southey, M. P. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner, ''Bayes' bluff: Opponent modelling in poker,'' 2012, *arXiv:1207.1411*.

[64] N. Mizukami and Y. Tsuruoka, "Building a computer Mahjong player based on Monte Carlo simulation and opponent models," in *Prof. CIG*, 2015, pp. 275–283.

[65] J. Walton-Rivers, P. R. Williams, R. Bartle, D. Perez-Liebana, and S. M. Lucas, "Evaluating and modelling Hanabi-playing agents," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Donostia, Spain, Jun. 2017, pp. 1382–1389.

[66] J. Goodman and S. Lucas, "Does it matter how well I know what you're thinking? Opponent modelling in an RTS game," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Glasgow, U.K., Jul. 2020, pp. 1–8.

[67] R. S. Sutton and A. G. Barto, "Reinforcement learning," in *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018, ch. 1, pp. 1–4.

[68] A. G. Barto, P. S. Thomas, and R. S. Sutton, "Some recent applications of reinforcement learning," in *Proc. Yale Workshop Adapt. Learn. Syst.*, New Haven, CT, USA, 2017, pp. 1–13.

[69] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *Int. J. Robot. Res.*, vol. 37, nos. 4–5, pp. 405–420, Apr. 2018.

[70] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. CoRL*, Zürich, Switzerland, 2018, pp. 651–673.

[71] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: Lessons we have learned," *Int. J. Robot. Res.*, vol. 40, nos. 4–5, pp. 698–721, Apr. 2021.

[72] M. M. Afsar, T. Crump, and B. Far, "Reinforcement learning based recommender systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–38, Dec. 2022.

[73] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022.

[74] N. P. Farazi, B. Zou, T. Ahamed, and L. Barua, "Deep reinforcement learning in transportation research: A review," *Transp. Res. Interdiscipl. Perspect.*, vol. 11, Sep. 2021, Art. no. 100425.

[75] A. Haydari and Y. Yilmaz, "Deep reinforcement learning for intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 1, pp. 11–32, Jan. 2022.

[76] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Proc. NeurIPS*, Montréal, QC, Canada, 2018, pp. 1–15.

[77] L. Wang, Z. Pan, and J. Wang, "A review of reinforcement learning based intelligent optimization for manufacturing scheduling," *Complex Syst. Model. Simul.*, vol. 1, no. 4, pp. 257–270, Dec. 2021.

[78] J. Lee and M. Mitici, "Deep reinforcement learning for predictive aircraft maintenance using probabilistic remaining-useful-life prognostics," *Rel. Eng. Syst. Saf.*, vol. 230, Feb. 2023, Art. no. 108908.

[79] A. H. Ganesh and B. Xu, "A review of reinforcement learning based energy management systems for electrified powertrains: Progress, challenge, and potential solution," *Renew. Sustain. Energy Rev.*, vol. 154, Feb. 2022, Art. no. 111833.

[80] E. Foruzan, L. Soh, and S. Asgarpoor, "Reinforcement learning approach for optimal distributed energy management in a microgrid," *IEEE Trans. Power Syst.*, vol. 33, no. 5, pp. 5749–5758, Sep. 2018.

[81] A. Namdari, M. A. Samani, and T. S. Durrani, "Lithium-ion battery prognostics through reinforcement learning based on entropy measures," *Algorithms*, vol. 15, no. 11, p. 393, Oct. 2022.

[82] A. Namdari and Z. S. Li, "An entropy-based approach for modeling lithium-ion battery capacity fade," in *Proc. Annu. Rel. Maintainability Symp. (RAMS)*, Springs, CA, USA, Jan. 2020, pp. 1–7.

[83] A. Namdari and Z. S. Li, "A multiscale entropy-based long short term memory model for lithium-ion battery prognostics," in *Proc. IEEE Int. Conf. Prognostics Health Manage. (ICPHM)*, Detroit, MI, USA, Jun. 2021, pp. 1–6.

[84] D. Zhang, X. Han, and C. Deng, "Review on the research and practice of deep learning and reinforcement learning in smart grids," *CSEE J. Power Energy Syst.*, vol. 4, no. 3, pp. 362–370, Sep. 2018.

[85] R. Lu and S. H. Hong, "Incentive-based demand response for smart grid with reinforcement learning and deep neural network," *Appl. Energy*, vol. 236, pp. 937–949, Feb. 2019.

[86] Y. A. Sekhavat, "MPRL: Multiple-periodic reinforcement learning for difficulty adjustment in rehabilitation games," in *Proc. SeGAH*, 2017, pp. 1–7.

[87] Y. Shin, J. Kim, K. Jin, and Y. B. Kim, "Playtesting in match 3 game using strategic plays via reinforcement learning," *IEEE Access*, vol. 8, pp. 51593–51600, 2020.

[88] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[89] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," *Artif. Intell. Rev.*, vol. 55, no. 2, pp. 895–943, Feb. 2022, doi: 10.1007/s10462-021-09996-w.

[90] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[91] Á. López-Cardona, G. Bernárdez, P. Barlet-Ros, and A. Cabellos-Aparicio, "Proximal policy optimization with graph neural networks for optimal power flow," 2022, *arXiv:2212.12470*.

[92] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," 2016, *arXiv:1611.01224*.

[93] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proc. ICML*, 2015, pp. 1889–1897.

[94] Board Game Geek. *Ticket to Ride: New York*. Accessed: May 15, 2023. [Online]. Available: https://boardgamegeek.com/boardgame/253284/ticket-ride-new-york

[95] C. E. Shannon, "XXII. Programming a computer for playing chess," *London, Edinburgh, Dublin Phil. Mag. J. Sci.*, vol. 41, no. 314, pp. 256–275, Mar. 1950.

[96] J. C. Santamaria, R. S. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adapt. Behav.*, vol. 6, no. 2, pp. 163–217, Sep. 1997.

[97] J. B. Pollack and A. D. Blair, "Co-evolution in the successful learning of backgammon strategy," *Mach. Learn.*, vol. 32, no. 3, pp. 225–240, 1998, doi: 10.1023/A:1007417214905.

[98] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 12348–12355, 2021.

[99] Stable Baselines3. *Docs—Reliable Reinforcement Learning Implementations*. Accessed: Feb. 21, 2023. [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/

[100] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," 2020, *arXiv:2006.14171*.

[101] O. Marom and B. Rosman, "Belief reward shaping in reinforcement learning," in *Proc. AAAI/IAAI/EAAI*, New Orleans, LA, USA, 2018, pp. 3762–3769.

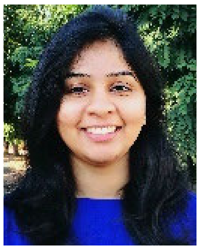
[102] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," 2015, *arXiv:1507.06527*.

SHUO YANG received the Ph.D. degree in computer science from the University of New South Wales, Australia, in 2022. He is currently a Postdoctoral Research Associate with the School of Engineering and Information Technology, University of New South Wales. His research interests include trustworthy AI, multi-agent systems, machine learning, human–machine interaction, and serious games.

MICHAEL BARLOW received the Ph.D. degree in computer science from the University of New South Wales, Australia, in 1991. Upon completion of the Ph.D. degree, he joined the University of Queensland, Australia, as a Postdoctoral Researcher, and thereafter Nippon Telegraph and Telephone's Human Communication Laboratories, Japan. In 1996, he joined the University of New South Wales, Canberra, where he is currently a Professor and the Head of the School of Engineering and IT (acting). His research interests include simulation, virtual environments, machine learning, serious games, and human–computer interaction.

THOMAS TOWNSEND received the B.Soc.Sc., B.Comm. (Hons), and Ph.D. degrees in information systems, in 2001, 2002, and 2020, respectively. He has more than 20 years of IT industry experience in operational, leadership, advisory, and consulting roles across higher education and government. He is currently based in the ACT, Australia. He is also a Senior Lecturer in cyber security with the University of New South Wales.

XUEJIE LIU received the bachelor's degree in telecommunication engineering and international economy and trade from Ludong University, in 2011, the Master of Science degree in acoustics from the South China University of Technology, in 2014, and the Ph.D. degree from the University of New South Wales, Canberra, Australia, in 2019. She is currently a Research Associate with the School of Engineering and Information Technology, University of New South Wales. She is also working on the brain–computer interfaces, electroencephalographic signal processing, and reinforcement learning agent in games.



DILINI SAMARASINGHE received the Ph.D. degree in computer science from the University of New South Wales, Canberra, Australia, in 2021. She is currently a Postdoctoral Research Associate with the School of Engineering and Information Technology, University of New South Wales. Her research interests include artificial intelligence, serious games, autonomous agent systems, and machine learning.

ERANDI LAKSHIKA (Senior Member, IEEE) received the B.Sc. degree (Hons.) in computer science from the University of Colombo, Sri Lanka, and the Ph.D. degree in computer science from the University of New South Wales, Canberra (UNSW Canberra), in 2014. In 2009, she joined the University of Colombo School of Computing, as an Assistant Lecturer. She is currently a Senior Lecturer with UNSW Canberra. Her research interests include human–computer interfaces, multi-agent systems, computational intelligence, multi-objective optimization, serious games, and games for health.

GLENNN MOY received the Bachelor of Science degree (Hons.) from The University of Queensland and the Ph.D. degree in theoretical physics from The Australian National University. He is currently with the Australian Defence Science and Technology Group (DSTG), where he has developed a range of analytical tools, including in areas of planning, logistics, and capability modeling. His current research interests include artificial intelligence and machine learning, where he has investigated the role of deep learning techniques to enhance decision-making.

TIMOTHY LYNAR received the Ph.D. degree from The University of Newcastle, Australia, in 2011. He was with IBM Research for 7.5 years, where he was a Research Staff Member and a Master's Inventor, before joining the University of New South Wales, in 2019. He has a background in simulation, modeling, and distributed computing, including cloud and the IoT systems. His primary research interest includes the application of machine learning to cyber security.

BENJAMIN TURNBULL (Senior Member, IEEE) was with the U.S. Naval Research Laboratory, the Australian Department of Defence, private industry, and the Royal Military College, Canada. He researches and teaches in the area of cybersecurity. Before he transitioned to academia, he was with the Defence Science and Technology Group. He has been working in cybersecurity and related fields for 19 years. He is currently an Assistant Professor with the University of New South Wales, Canberra.