Reinforcement Learning in Hanabi

Andrea Lisi
Department of Computer Science
Università di Pisa
andrea.lisi@phd.unipi.it

Abstract

Real world situations typically require cooperation among individuals, and this is reflected in board games with a cooperative gameplay. Given the success of Machine Learning algorithms beating international champions of Go and similar games, a new challenge consists in improving learning agents so that they are able to succeed together in a task only with cooperation. In 2018 the game of Hanabi was chosen as a benchmark for this task because it provides a cooperative challenge with partial observability and limited communication. This report lists how the scientific community reacted to this challenge proposing methods to play the game of Hanabi.

1 Introduction

A board game is a type of game that humans play since centuries, and evolved through the time until they have the shape we are familiar with. For example, the ancestor of the game of Chess is Chaturanga from the 7th century India. Typically, board games consist of a board and physical pieces that are placed, moved and removed by the players. The number of players vary between 1 in solo games like Solitaire, and several in party games like Taboo. Many games allow the players to compete with each other, free for all or in teams, pushing a player to choose their moves in such a way they perform better than the adversaries in order to win the game. Usually, random factors improve the variety from game to game helping their re-playability, but it may unintentionally help a player to win: an example is Risk. Others have the same setup between each game and no random factors are involved, like in Chess. Very popular are also cooperative games, i.e. games where all the players try to defeat the game, like in Pandemic, or try to maximise their score, like in Hanabi. Semi-cooperative games exist as well: all the players play together but not all of them win, like in Dead of Winter; a player is in reality an hidden traitor and cooperates until they need to, like Betrayal at the House on the Hill.

Board games stimulate the players to play the game as best as they can. As a consequence in the digital time, the development of computer programs able to play such games as well as humans became a priority for many companies. Indeed, artificial intelligence is vastly used in computer games, also when they replicate a physical board game. With the growth of popularity of machine learning algorithms the research community put effort in the development of models able to compete with world champions in different board games. Iconic is the example of AlphaGo, a machine learning agent developed by Google DeepMind able to beat international champions at the game of Go.

The growing popularity of machine learning techniques stimulated the research community to investigate new approaches to tackle a problem. In this report we focus on Reinforcement Learning (RL) [19], a paradigm alternative to supervised and unsupervised learning based on the maximisation of a reward that is accumulated after picking a sequence of actions in an environment that can be observed, fully or partially, by the agent.

Given the success of learning agents in zero-sum board games like Go [10], i.e. games that a loss to a player has the same weight as the gain to the adversary[1], a new frontier in the development of learning agents consists in facing the problem of cooperation, i.e. allow a learning agent to cooperate with a human in a task and succeed, or fail, together. For this reason, cooperative games build good environments for the development of methods to be used by learning agents, and then generalize such methods to tackle real world problems [6, 15]. In this report we focus the attention on methods published in literature based on Reinforcement Learning to learn to play a famous cooperative game called Hanabi. RL methods have been successfully used to build agents to play zero-sum games [17] and non, like Settlers of Catan [23]. The game of Hanabi pushes the research a step further.

2 The game of Hanabi

Hanabi [4] is a cooperative card game for 2-5 players created by Antoine Bauza in 2010 and winner of the prestigious "Spiel des Jahres" (game of the year) in 2013. The cards represent fireworks[2] of 5 different colors, blue, red, yellow, white and green, and numbers, from 1 to 5. For each color, the number 1 has 3 copies, numbers 2, 3 and 4 have 2 copies each, and the number 5 has only 1 copy. Therefore the total number of cards are 50. The players have also 8 hint tokens and 3 life tokens.

The rules of Hanabi are simple: the players need to build 5 stacks, one for each color, and play the cards in an ascending order, from 1 to 5. At the end of the game the points scored by the players will be the sum of the topmost card on each stack. Therefore, with 5 colors and 5 numbers the highest score possible is 25. The players start the game with 4 or 5 cards in their hand (for 2-3 players and 4-5 players respectively), but faced to the other players: a player sees the cards of the other players, but not their own. During their turn a player has 3 possible actions:

1. *give an hint*: the current player picks a number or a color, and tells to one other player *all* the cards in their hand of that color or that number[3]; this action costs 1 hint token, and it is not possible to do this action when no hint tokens are available;
2. *discard a card*: the current player can discard a card from their hand to recover an hint token and draw a new card. In this way an hint token can be spent again by another player;
3. *play a card*: the current player can play one card of color C and number N from their hand to one of the stacks (it is not required to specify which stack): if $N = 1$, the player can put down that card if a 1 of color C it has not been played already; otherwise, that card can be played only if the topmost card of color C shows the number $N - 1$. If a card does not fit in the two conditions above, the player discards that card, does not gain any hint token and the players lose a life token: this means the players can do at most 3 mistakes. In any case, the player draws a new card.

After a player performed an action passes their turn to the next player. The game ends when the players lose all the life tokens, scoring 0 points, or when the stack of cards to draw from is over (the players have time for a last round). An additional minor, but important, rule is when a 5 has been played correctly, the players gain an hint token.
To give an idea whether an agent performs well in the game of Hanabi, this is the list of score intervals given by the game: 0-5 horrible, 6-10 mediocre, 11-15 honourable, 16-20 excellent, 21-24 amazing, 25 legendary. They represent the appreciation of the crowd to the fireworks.

As a result, Hanabi is a cooperative game where a player has incomplete, or imperfect, information of the environment, i.e. their cards, and with a limited communication protocol is present, i.e. the hint

[1] For example, if I get the biggest piece of cake then my cake gain is equal to the cake lost by my friend.

[2] Hanabi means firework in Japanese.

[3] For example, if I want to tell to the player in the figure he has a white one, I can tell him only "These two cards are a one" including the yellow one that has already been played, or "These cards are white", including the two, not playable.

tokens. Since the players need to cooperate together, communication is key [6] but it is limited in both expressiveness and amount (because giving an hint costs a token). Moreover, the structure of an hint makes it subjected of interpretation: two human players can understand a different intention from the same hint. Moreover, communication between players is prone to human errors, like changing the tone of voice, use the look, react to a move of another player. The authors in [1] show that scoring 25 points in the game of Hanabi playing like a solitaire where you can look at your cards is NP-Complete. Considering all of these factors, Hanabi creates a good challenging environment to evaluate cooperative learning agents.

2.1 The Hanabi AI competition

The Conference of Computational Intelligence and Games (CIG) held in 2018 proposed a competition to create AI agents able to play Hanabi in two settings: with replicated, and non, teammates respectively called *self-play* and *ad-hoc*. In self-play an agent plays with copies of itself, specifically agents with the same training, meanwhile in ad-hoc an agent plays with other agents, or its copies but trained independently. The game provides an interesting challenge to learning agents due to the limited knowledge of the environment and the need to cooperate in order to effectively play the game: real world problems are mostly tackled in cooperation [15]. An agent needs to understand *why* a certain hint was given to it or to another agent: this is "easy" to understand if an agent plays with an equivalent one that follow the same conventions, and that is why the competition proposes also random team match ups. This reflects real human players who do not know each other and approach the game differently. The details of the competition can be found in [22].

3 Background and State of the art

This section briefly introduces the basics of RL and why its is a suitable paradigm for Hanabi. Next, it presents the learning agents participating to the CIG 2018 competition [22] and the RL agents that have been proposed in literature afterwards. Finally, this section presents a few rule-based bots for Hanabi that will be compared with the learning agents when presenting the game results in the next section. A rule is a function that given the game state and the active player returns a legal action to do [5].

3.1 Reinforcement Learning

The paradigm of RL inherits from the Markov Decision Processes (MDP) that is formalized as a tuple (S, A, P, R, γ), where S is the set of states our agent can be in, A is the set of actions the agent can pick, P is the state transition matrix that tells the probability of going to state s' picking action a from state s, i.e. $P(S_{i+1} = s'|S_i = s, A = a)$ where i is the current timestamp, R is the function computing the expected reward from picking action a from state s, and $\gamma \in [0, 1]$ is the discount factor, i.e. a factor that favours long term rewards over short term ones if γ tends to 1, the opposite if γ tends to 0. The goal of a RL agent is to learn a policy π, eventually probabilistic, telling the agent which action to pick in a particular state so that the future reward is maximised: to the future reward is applied the discount factor γ and it is know as "expected return". The concept of value function is used to evaluate how good being a certain state, or picking a certain action, are. Therefore, the optimal policy an agent wishes to learn is the one maximising the value function. However, optimizing such function is feasible only for MDPs with a limited number of states, and sometimes such states are not even available to the agents: in this latter case, the agent needs to learn by exploring i.e. picking actions and experiencing the reward. A taxanomy of RL methods can be found in [21].

RL approaches can be divided into three groups: actor-only, critic-only and actor-critic. Actor-only methods use direct policy optimization methods with the advantage of generating a spectrum of continuous action, but leading to slow learning rate due to high variance in the estimates of the gradient. Critic-only methods using Temporal Difference learning have lower variance, but at each state they require to find an optimization procedure leading to the best expected return that might be computationally intensive, therefore they rely on a discretization of the continuous action space but lowering the chance of finding the optimal value. Actor-critic methods combine the advantages of actor and critic only methods. The critic estimates the expected returns, helping the actor to update its gradient with lower variance, speeding up the process, but at the cost of higher bias at the beginning of

the process. Overall, actor-critic methods have good convergence properties. A survey of actor-critic methods can be found in [13].

In a multi-agent RL model an agent learns in a environment shared with other agents, and knowing that these agent exist and interact as well. Typically the environment is not fully observable, and because of that the challenge consists in adopting a valid communication protocol among the agents while learning the policy (a suitable real world scenario). Each agent independently takes an action sampled from the policy, but the next state of the environment is conditioned by the joint action taken by each agent. In a fully cooperative settings each agent receives a team reward that depends on the last joint action. The centralized training and decentralized execution approach can be followed, meaning that during training each agent can exchange any information, but the policies need to be compatible with an execution where information cannot be freely exchanged.

The game of Hanabi is a suitable benchmark for RL models, especially multi-agents. The environment is partially observable, therefore generating a Partially Observable MDP (POMDP) where an agent cannot query (at least partially) the underlying state: some of the cards are unknown to the agent. The game is fully cooperative and the players do not have "freedom of speech", but rather a restricted communication protocol spending hint tokens that are limited but refreshable. Therefore, the goal of a single agent it to optimize the usage of the hint tokens so that the other agents can play or discard a card safely. The tricky part is to balance of short term and long term hints, because playing cards allows the game to continue, but warning players of high numbered cards, especially 5s who give an hint when played, is crucial to reach a good score. Moreover, actions are subjected to interpretation, and gives information both "why that action was chosen" and "why not *this* other action was chosen".

3.2 Learning agents for Hanabi

Both of the tracks of the 2018 competition were won by agents implementing an extension of the Information Set-Monte Carlo Tree Search (IS-MCTS): in this report we focus on the agent winning the self-play track and presented in [12]. At the current state, standard IS-MCTS search over the information (game) tree selecting each time an action, simulate the progress of the game with that action, compute the score and back-propagate it up to the tree. The model has a time limit to perform these iterations, and at the end it picks the action that gave the maximum score. Regardless of the time limit, IS-MCTS performs poorly on Hanabi, but not in other games, due to the lack of complete information. The authors improve it avoid leaking hidden information with a model called Re-determinizing IS-MCTS (RIS-MCTS), i.e. sampling a possible set of hidden information (the player's cards) and explore the tree assuming a state with the sampled information: this sample is built independently from player to player, and is produced at every node of the game tree.

Another competing agent was developed and proposed in [5]. In this work the authors propose an evolutionary agent based on genetic algorithms to build the rules, order them and choose the next move. In contrast, rule-based bots typically have a fixed sized set of rules hard-coded by humans and they search over them to find the best move to do at a certain point in the game. The main goal of the algorithm is to build a good ordering for the generated rules. The algorithm generates chromosomes[4], randomly initialized, of size the number of rules in a particular experiment, with 50 and 72 being the most significant sizes, and improving them generation after generation with two fitness functions: self-play and ad-hoc game performances.

After the competition a new branch of research opened to teach learning agents to play a game like Hanabi. We list here a few examples involving RL agents.

Actor Critic Hanabi Agent The authors in [3] provide a learning environment for Hanabi, and propose an experimental framework for the evaluation of the learning agents. In their work, the authors present ACHA, Actor Critic Hanabi Agent, an agent based on actor critic RL as the name suggests. The paper does not present many details on the model, but rather lists the features and parameters that have been set to implement it. ACHA runs 100 actors generating experience and parameter sharing across the players, including the player ID to allow a level of specialization between the agents: this increases the learning speed. The observations are processed by a multi layered perceptron, 256 units with ReLu activations, and then by 2-layer long short term memory network

[4] A chromosome in a genetic algorithm is a set of parameters identifying a possible solution.

with 256 units per layer. A softmax and a linear model are applied to the output to generate the policy and baseline respectively.

Since the game reflects the *theory of mind* [2], i.e. the ability to be informative when performing an action, the following works, BAD and SAD, are based on Bayesian reasoning, performing a Bayesian update after each observed action. A Bayesian update consists in updating the probability (posterior probability) of an hypothesis (prior probability) as soon we have more information about it using the Bayes theorem. Given an hypothesis H and an evidence E as two events and a probability function $P()$, the Bayes theorem states that:

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E)} \tag{1}$$

That means, the probability of the hypothesis H *posterior* to having observed the evidence E, is equal to the probability of H *prior* to having observed E ($P(H)$) multiplied by the *likelihood* of E happening given H ($P(E|H)$), i.e. the degree to which the hypothesis predicts E, and divided by the probability of having that evidence E [20].

Bayesian Action Decoder In [9] the authors present Bayesian Action Decoder (BAD), a multi-agent learning method based on Bayesian updates that condition the current policy of the playing agent, and therefore the actions taken in the environment. This model is developed to discover both the communication protocol and the policy in a cooperative and partially observable environment. The setup is a POMDP. The Markov state consists of a set of a discrete features both public and private. The public features f_t^{pub} are known by all the agents (e.g. the played and discarded cards), while the private ones f_t^{pri} are known by at least one agent but not all of them (e.g. the cards in hand). The agent is powered with the concept of *public belief* that needs to be adapted from single-agent to multi-agent environment (our personal believes may be incorrect since other agents have private information we do not have access to). The authors build the public belief MDP (PuB-MDP), based on another previous work, adapted so that it represents the posterior over all the private state features given the public ones, i.e. $B_t = P(f_t^{pri}|f_{\leq t}^{pub})$ where t is the current timestamp and $\leq t$ indicates the history. The authors include a third party public-agent that observes only public features and belief, and builds a partial policy $\hat{\pi}$ at each timestamp t based on f_t^{pub} and B_t: $\hat{\pi}$ maps private observations to environment actions. Each acting agent a can query this policy with their private observation f_t^a and selects an action $u_t^a = \hat{\pi}(f_t^a)$. After analysing the outcome, the public agent updates the public belief to the next timestamp B_{t+1}. Instead of keeping all the information in each agent, the authors split the public observation into this external agent so that the public information are consistent among all the playing agents. The public belief, that we remark being the probability of a particular private feature at time t (our hypothesis H from Equation 1) given the history of all the public observations (our evidence E from Equation 1), is then updated following the Bayes theorem:

$$P(f_t^a|u_t^a, B_t, f_t^{pub}, \hat{\pi}) = \frac{P(u_t^a|f_t^a, \hat{\pi}) \times P(f_t^a|B_t, f_t^{pub})}{P(u_t^a|B_t, f_t^{pub}, \hat{\pi})} \tag{2}$$

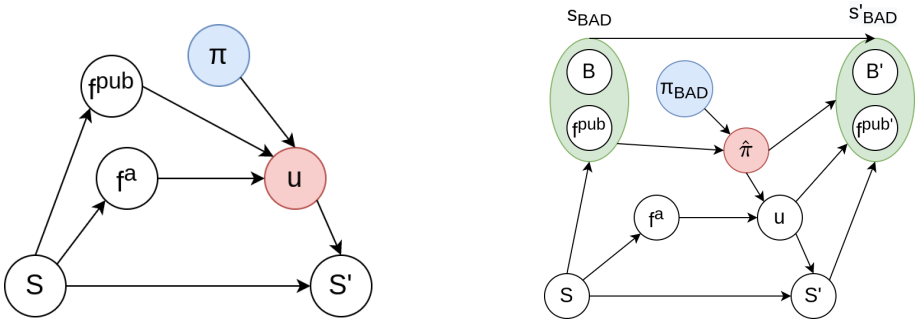
This equation defines the PuB-MDP, and its differences with an MDP are graphically represented in Figure 1 where the state is $s_{BAD} = \{B, f^{pub}\}$, the action space is the set of deterministic partial policies mapping private observations to environment actions, and the transition function is $P(s'_{BAD}|s_{BAD}, \hat{\pi})$, depending not only on the selected action u_t^a by agent a, but on the actions specified by $\hat{\pi}$, because the new public belief B' is computed via an update that conditions on $\hat{\pi}$. Finally, the reward function is computed over all the private features as follows:

$$r_{BAD}(S_{BAD}, \hat{\pi}) = \sum_{f^{pri}} B(f^{pri}) r(s, \hat{\pi}(f^{pri})) \tag{3}$$

In Figure 1(b) π_{BAD} represents the policy of the public agent, therefore updated only by the public information, and for each public state it must sample a partial policy over the deterministic partial policies. Since this space can be exponential, the authors limit it assuming a distribution across $\hat{\pi}$ factorizing across private observations, i.e. for all $\hat{\pi}$

$$\pi_{BAD}(\hat{\pi}|B_t, f^{pub}) = \prod_{f^a} \pi_{BAD}(\hat{\pi}(f^a)|B_t, f^{pub}, f^a) \tag{4}$$

This update must be public so that all the agents are able to get the same results, and the sampled $\hat{\pi}$ must be deterministic for all the agents.



(a) MDP (b) Pub-MDP

Figure 1: Difference between the MDP and Pub-MDP proposed in [9]

Simplified Action Decoder In [15] the authors present a multi-agent RL method called Simplified Action Decoder (SAD) to effectively play Hanabi in self-play mode. This method, as stated by the authors, achieves a goal similar to BAD, i.e. implementing an effective exploratory protocol, but trying to limit the computational burden that BAD needs to sample the partial policy (see Equation 4), and to avoid local optima issues. This method relies on the centralized training and decentralized execution. One author of BAD is involved in SAD as well, this might be the reason the models have a similar mindset. Like in BAD, the setup is a POMDP. During training each agent takes two actions: an informative greedy one, that is *not* executed by the environment but observed by the team mates, and an exploratory action, that gets executed by the environment and observed by the team mates as well. After each action the agent performs the correspondent Bayesian update described by equations 5 and 6 respectively. If the agents stop exploring, the greedy actions will be executed by the environment. Given the assumption that each agent a can observe the greedy action at each timestamp t during training, and perform a Bayesian update that is structured as follows:

$$P(\tau_t|\tau_t^a) = \frac{\mathbf{I}(u^*(\tau_t, u^*))B(\tau_t)}{\sum_{\tau'}\mathbf{I}(u^*(\tau_t, u^*))B(\tau')} \tag{5}$$

where u^* is the greedy action, τ_t is the state-action sequence at timestamp t, $\tau = \{s_0, u_0, r_1, s_1..., r_T, s_T\}$ is the action-state history with s_i the state, u_i the action and r_i the reward, and τ^a is the action-observation history of an agent a, i.e. $\{o_0^a, u_0^a, r_1, s_1..., r_T, o_T^a\}$ with o_i^a the observation of a at timestamp t. Finally, $B(\tau_t)$ represents the belief about the state-action history of the world given the action-observation history of the agent a, i.e. $B(\tau_t) = P(\tau_t|\tau_t^a)$. We remark that this is not the only update that an agent performs during training, but this in support to the update performed after the exploratory action: this is compatible with the centralized training regime. The exploratory action is based on an ϵ-greedy exploration scheme, meaning that the agent has a probability ϵ of taking an exploratory action, a greedy action otherwise, and its Bayesian update is structured as follows:

$$P(\tau_t|\tau_t^a, u_t^a) = \frac{((1-\epsilon)\mathbf{I}(u^*(\tau_t), u_t^a) + \frac{\epsilon}{|U|})B(\tau_t)}{\sum_{\tau'}((1-\epsilon)\mathbf{I}(u^*(\tau'), u_t^{a'}) + \frac{\epsilon}{|U|})B(\tau')} \tag{6}$$

where a' is a team mate of a. During the execution the agents will follow a fully greedy approach setting the parameter $\epsilon = 0$, compliant with the decentralized execution regime.

Given the importance of search algorithms in zero-sum games[5], the authors propose an enhancement to SAD applying search algorithms with the goal to improve an arbitrary agreed-upon (labeled as "blueprint") policy [16] called Search for Partially Observable Team of Agents (SPARTA). In the *single-agent* search, only one agent a performs an online search while the others follow the blueprint policy (and a knows it), it maintains a belief distribution over the possible search paths based on its action-observation history and the known blueprint policy of the other agents: each time a *observes* an action it, updates its belief with a Bayesian update; each time a *has to performs* an action it estimates the expected value of its value function via Monte Carlo rollouts for each action a can take

[5]An algorithm used to solve a search problem, like finding the minimum in a tree. In games that means exploring the possible states that game can end up performing a move (also know as "a ply").

Agent	2-players	3-players	4-players	5-players	Ref
RIS-MCTS	20.5	22.0	21.3	20.0	[12]
Genetic	19.61	19.68	19.11	17.87	[5]
ACHA	22.73	20.24	21.57	16.80	[3]
BAD	24.17	-	-	-	[9]
SAD	24.01	23.93	23.81	23.01	[15]
SAD - search	24.53	24.62	24.49	23.91	[16]
WTFWThat	19.45	24.20	24.833	24.89	[3]
Smartbot	22.99	23.12	22.19	20.25	[3]
Fireflower	22.56	21.05	21.78	-	[3]

Table 1: A summary of the average scores for each model in self-play mode. The **Ref** column includes the sources of the scores.

(only for the **next** action, otherwise it would become too expensive). In the *multi-agent* search all the agents agree both on the blueprint policy and the search procedure to use. When a conducts a search when performing an action, the other agents conduct the same search computing a's resulting policy accordingly, but the other agents do not know a's private observations so they compute a's policy for every possible action-observation history that a might be in on the common knowledge observations. The size of the history can be very large (10 million in 2-players Hanabi), but the amount of positive-probability histories can be much smaller: anyway, all the agents agree that if that space is larger than m at timestamp t, then a does not perform any search and plays according to the blueprint policy (so neither a nor the other agents conduct any search). According to the paper, this enhancement improves all the average scores of simple SAD.

3.3 Rule-based bots for Hanabi

As a term of comparison, the learning agents have been compared to rule-based Hanabi bots that can be openly found online. A rule-based agent does not learn a strategy after a training process, but is rather built on top of a predefined set of rules allowing the bot to pick a move in a certain situation. These bots follow custom a-priori conventions so that an hint may give more than the information it is allowed to (for example, when I tell you a card is red or yellow colors, if you see you cannot play them in this turn it means they are to discard), but it is useful only if the receiver knows that conventions as well. Therefore, these bots should not perform well in ad-hoc games.

Smartbot [18] combines personal view of the game plus public knowledge (e.g. played and discarded cards) so that Smartbot understands in some points in the game if a cards it has is safe to discard / play in order to spare an hint. Similarly, Hatbot [7] utilizes a predefined protocol when giving hints, so that an hint action can give multiple information and possibly to the other players too. However, the original proposal of Hatbot is restricted to five players games, so an extension has been developed, called WTFWThat [11], that allows games from two to five players. Finally, Fireflower [8] implements a set of human conventions and computes, for a possible action, the probability distribution of the responds that the other player will do so that the action to be chosen will maximise the expected result. The goal of Fireflower is to maximise the win rate, and not the average score of each game.

4 Winning performances and experimental results

This section presents the average scores the aforementioned approaches are able to reach after playing the game several times. Some works compare themselves with the other approaches, and the resulting score can differentiate because the experimental setups may differ. For that reason we show, for each model introduced in Section 3.2, the scores provided by their authors. Table 1 show a summary of the scores. The first group includes the two models competing the 2018 Hanabi competition, the second group includes the RL-based models that have been developed as a research challenge after the competition, and the last group include the rule-based Hanabi bots. If a work has more setups, like [12], leading to more score lines, the table shows only one of them.

In their work [12] the authors show many experimental configurations for their RIS-MCTS agent when playing Hanabi, like the amount of time the agent has to make a move, the simulation policy used by

MCTS and the usage or not of a convention called "playable now" that, if enabled, discourages some hints to the cards that are not playable at the moment. For example, with a MCTS random simulation policy, no "playable now" convention and 1 or 10 seconds the agent reaches scores between 17.0 and 18.9 between 2 and 5 players (higher scores with 3 and 4 players). The addition of the "playable now" convention increases the average score up to 20.6 with 2 players, and almost 20 with 3-4 players and 18.8 with 5 players. The usage of a non-random simulation policy reaches comparable scores as the "playable now" convention, and the agent reaches its best results when both of them are applied, reaching 22.0 and 21.3 with 3 and 4 players, and 20.5 and 20.0 with 2 and 5.

The work who reached the second place at the competition, described in [5], reaches an average scores that goes from 17.02 with 5 players, up to 19.35 with 2 players. By applying certain conventions like "play when hinted", and training the agents specializing on the number of players, the authors succeeded to improve the average score of 1 point. The authors also present the performance on the ad-hoc playing mode, which go from an average score of 12.21 with 2 players, and 10.55 with 5. Applying the same enhancements as in the self-play mode, the average scores improve only by 0.2 - 0.3 points.

In [3] the authors compare their agent, ACHA, with BAD and the rule-based bots. The agent scores in average between 20 and 22 points in a 2-4 players setting, with a small percentage of perfect games (from 11% with 2 players and 1.1% with 3 players), and scoring only 16.8 points with no perfect games in a 5 players setting. The authors also test ACHA in an ad-hoc mode, i.e. the agent playing with other agents. The chosen agent is called Rainbow [14], a Deep Q Learning (DQN) agent. The authors pick the best 10 trained ACHA and the best trained Rainbow agents, and let them to play 2 and 4 players games with all the others and also themselves: therefore, the results include self-play trials. As expected, self play reach the average scores listed previously, but the performance of games among different agents have very bad scores, never reaching 5 points. With this experiment the auhtors want to emphasize the difficulty on training a learning agent that is able to adapt to different agents, and therefore different strategies.

The BAD agent [9], as introduced, is able to play only games with two players but outperforming all the previous agents. In their work the authors show that the agent is able to make a perfect game 58.6% of the times, and reaching an average score of 24.174. The SAD agent [15] reaches an high average score for all the combination of players, with a 24.01 for 2 players and between 23 and 23.93 with 3, 4 and 5 players. In the experiments shown by the authors, SAD is able to beat the average score of BAD, but not its win rate percentage (that we assume to mean "perfect game", otherwise the average score would be much lower). However, SAD improved with search [16] is able to reach higher average scores than SAD, and BAD also, for all the player settings: the scores shown in Table 1 are related to the single-agent search procedure.

The rule-based bots perform well, often better than the learning agents. WTFWThat seems to be the most performant one in [3], reaching an average score of 24.89 (amazing) in a 5 players setting with 91.5% of perfect games, 24.2 and 24.8 in 3 and 4 players settings respectively. It performs worst in a 2 players setting with an average score of 19.5. Smartbot instead performs better with lower players (about 23 points), but having a small percentage of perfect games, and worst with 5 players, scoring in average 20.3 points with almost 0% of perfect games. In contrast, FireFlower has in average a smaller score, but winning more perfect games: as an example, in a 4 players setting SmartBot has an average score of 22.19 with 2% of perfect games, meanwhile FireFlower has an average score of 21.78 with 26.5% of perfect games.

5 Conclusions

This report gives an overview of the learning agents that have been proposed in literature to play the game of Hanabi. Understanding how to address problems risen by games helps the development of solutions to similar problems in the real world, and Hanabi represents a challenge to the research community since an agent can observe the environment only partially and needs to cooperate through limited communication with others in order to succeed in a task, i.e. maximise the score. The agents proposed in literature so far are able to reach good scores at the game when playing with copies of themselves, but this does not happen when an agent plays with different ones that have different approaches and methodologies, and this situation reflects better the real world. Moreover, Hanabi includes a 6th rainbow color that it is all the colors at the same time, and not a 6th color: when

pointing cards of a given color the player must include also the rainbow cards if there is any. This adds a new level of complexity for the agent because a red card may be either red or rainbow.

References

[1] Jean-François Baffier et al. "Hanabi is np-hard, even for cheaters who look at their cards". In: *Theoretical Computer Science* 675 (2017), pp. 43–55.

[2] Chris L Baker et al. "Rational quantitative attribution of beliefs, desires and percepts in human mentalizing". In: *Nature Human Behaviour* 1.4 (2017), pp. 1–10.

[3] Nolan Bard et al. "The hanabi challenge: A new frontier for ai research". In: *Artificial Intelligence* 280 (2020), p. 103216.

[4] *BGG, Hanabi*. URL: `https://boardgamegeek.com/boardgame/98778/hanabi`. Accessed: 8 September 2020.

[5] Rodrigo Canaan et al. "Evolving agents for the hanabi 2018 cig competition". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2018, pp. 1–8.

[6] Rodrigo Canaan et al. "Towards game-based metrics for computational co-creativity". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2018, pp. 1–8.

[7] Christopher Cox et al. "How to make the perfect fireworks display: Two strategies for hanabi". In: *Mathematics Magazine* 88.5 (2015), pp. 323–336.

[8] *Fireflower, repository*. URL: `https://github.com/lightvector/fireflower`. Accessed: 8 September 2020.

[9] Jakob Foerster et al. "Bayesian action decoder for deep multi-agent reinforcement learning". In: *International Conference on Machine Learning*. 2019, pp. 1942–1951.

[10] Imran Ghory. "Reinforcement learning in board games". In: *Department of Computer Science, University of Bristol, Tech. Rep* 105 (2004).

[11] *Github, wuthefwasthat/hanabi.rs*. URL: `https://github.com/WuTheFWasThat/hanabi.rs`. Accessed: 8 September 2020.

[12] James Goodman. "Re-determinizing MCTS in Hanabi". In: *2019 IEEE Conference on Games (CoG)*. IEEE. 2019, pp. 1–8.

[13] Ivo Grondman et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[14] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[15] Hengyuan Hu and Jakob N Foerster. "Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1912.02288* (2019).

[16] Adam Lerer et al. "Improving Policies via Search in Cooperative Partially Observable Games." In: *AAAI*. 2020, pp. 7187–7194.

[17] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[18] *Smartbot, Hanabi bots repository*. URL: `https://github.com/Quuxplusone/Hanabi`. Accessed: 8 September 2020).

[19] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[20] Richard Swinburne. "Bayes' Theorem". In: (2004).

[21] *Taxanomy of RL methods*. URL: `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`. Accessed: 8 September 2020).

[22] Joseph Walton-Rivers, Piers R Williams, and Richard Bartle. "The 2018 hanabi competition". In: *2019 IEEE Conference on Games (CoG)*. IEEE. 2019, pp. 1–8.

[23] Konstantia Xenou, Georgios Chalkiadakis, and Stergos Afantenos. "Deep Reinforcement Learning in Strategic Board Game Environments". In: *European Conference on Multi-Agent Systems*. Springer. 2018, pp. 233–248.