Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2022) 30: 2303 – 2318
© TÜBİTAK
doi:10.55730/1300-0632.3940

Learning to play an imperfect information card game using reinforcement learning

Buğra Kaan DEMİRDÖVER, Ömer BAYKAL*, Ferda Nur ALPASLAN
Department of Computer Engineering, Faculty of Engineering, Middle East Technical University, Ankara, Turkey

Received: 21.10.2021 • **Accepted/Published Online:** 05.08.2022 • **Final Version:** 28.09.2022

Abstract: Artificial intelligence and machine learning are widely popular in many areas. One of the most popular ones is gaming. Games are perfect testbeds for machine learning and artificial intelligence with various scenarios and types. This study aims to develop a self-learning intelligent agent to play the Hearts game. Hearts is one of the most popular trick-taking card games around the world. It is an imperfect information card game. In addition to having a huge state space, Hearts offers many extra challenges due to its nature. In order to ease the development process, the agent developed in the scope of this study was divided into subagents such that each subagent was assigned a part of the game. The experiment results reveal that the developed agent can compete against some rule based Hearts agents and human Hearts players.

Key words: Artificial intelligence, machine learning, reinforcement learning, supervised learning, neural networks

1. Introduction

Games are perfect testbeds and one of the most important driving forces of the Machine Learning (ML) field. In this study, a self-learning intelligent gameplaying agent for the Hearts card game was developed. Hearts is one of the most popular card games around the world. Nevertheless, prior work could not achieve human-level gameplaying in Hearts. However, there is some work, [1], that achieved human-level performance in a simplified version of the game (i.e. a version without shooting the moon and passing). Hearts is an imperfect information trick-taking card game. Players in games with perfect information (e.g., chess and checkers) have access to the complete/perfect information, but players in an imperfect information game do not. In Hearts, each player's cards are hidden from the rest of the players. Hearts is played with four players, and winning tricks (i.e. taking cards) is usually, not always, penalized. The game has two phases: *passing phase* and *playing phase*. Each round in the game starts with the passing phase and continues with the playing phase. A detailed explanation of the game is given in Section 3.

Hearts is a challenging and interesting game to develop intelligent agents for because of three main reasons. First of all, the game has contradictory rules. In other words, many decisions that should be made by Hearts players include trade-offs. Second, the game includes different phases and processes. As stated before, the game includes two consecutive phases, which are passing and playing. Each phase also includes subprocesses. So, Hearts players should be very dynamic during the gameplay. Third, the game has a huge state space. There exist lots of possibilities at an arbitrary time of the game. Due to the complexity of the Hearts game, instead

*Correspondence: obaykal@ceng.metu.edu.tr of training a single intelligent agent, a system that comprises many intelligent *subagents* was developed. Each subagent in this system is responsible for one part of the game. All details of the subagents are given in Section 4. Basically, there are two types of subagents. The first one is the *Reinforcement Learning (RL) subagents* that select cards to pass or play. RL is used to approximate state-value functions, which are functions that estimate values of the game states. The state-value functions in this study were represented by Artificial Neural Networks (ANN). The second type of subagents is the *Supervised Learning (SL) subagents* that selects RL subagents to make the next game moves. The data needed to teach SL subagents were collected from the RL subagents. The SL subagents were also developed using ANNs.

The purpose of this study is to introduce a new system consisting of more than one intelligent agent which can learn the complete version of the Hearts at human level without removing the key aspects of the game such as shooting the moon and passing. By introducing more than one agent for different parts we managed to include all aspects of the Hearts. One other important contribution is comparing the human-level agents with real humans and proving they are indeed better than the average human player.

2. Related work

Machine learning is utilized in a wide range of areas like healthcare, translation, and network security [2]. One of the most important ones is games. Machine learning and games are in a mutually beneficial relationship. While games are perfect testbeds for machine learning applications, they, on the other hand, benefit from those applications. There are many studies related to games and machine learning in the literature. Studying games is popular among machine learning researchers because games are relatively easier to simulate, and they have challenging characteristics. Reinforcement learning is used in many game-related machine learning studies. Two popular game genres for machine learning studies are card and board games.

One of the earliest works for the card and board games are checkers learning agents developed by Arthur Samuel [3, 4]. They can be considered as the foundation of many later work. He used search-based methods and reached human-level performance in checkers. Apart from Samuel's works, another early work in literature is *TD-Gammon* by Gerald Tesauro [5, 6]. *TD-Gammon* is a backgammon agent trained using temporal difference learning [7]. In *TD-Gammon*, the nonlinear version of temporal difference learning (namely TD(λ)) was used with a multilayer neural network. *TD-Gammon* achieved to compete and eventually to win against the world backgammon champions. In [8], a chess-playing agent, called *Deep Blue*, was introduced. *Deep Blue* achieved to beat the world chess champion Garry Kasparov in 1997 [9]. In this work, a large dataset consisting of games of master chess players was used along with handcrafted features for the chess openings and endings. Another, also more recent, research is on the game Go. *AlphaGo* [10], developed by Google DeepMind, beat the world Go champion in all five matches. For *AlphaGo*, a combination of the Monte Carlo tree search algorithm and reinforcement learning was applied. Later, in *AlphaGo Zero* [11], *AlphaGo* was improved by removing the Monte Carlo tree search and the human factors from the learning process so that the *AlphaGo* became its own teacher. The learning was done with reinforcement learning only. *AlphaGo Zero* was proven to be better than *AlphaGo*: in hundred matches between these two agents, *AlphaGo Zero* won all hundred ones. Later in *AlphaZero* [12], the methods used in *AlphaGo Zero* were generalized so that the agent can learn more games simultaneously. *AlphaZero* was trained without any domain knowledge except the rules. It achieved superhuman performance in the games of chess, shogi, and Go. In more recent work, *MuZero* [13] managed to learn the same games even without the knowledge of game rules. In a study from 2019 [14], researchers present a survey of artificial intelligence for card games and apply MCTS to the Swiss card game Jass and get promising results on the way to develop a strong AI playing Jass. In a recent study, [15], Batak, a card game popular in Turkey similar to Spades, was studied. In that study, reinforcement learning agents were developed using Monte Carlo estimation of state-values. The study also utilizes neural networks for function approximation. In a study from 2018 [16], Charlesworth introduced a new simulated environment for the game of "Big Two", which is suitable for multi-agent reinforcement learning algorithm applications. He also successfully trained a neural network purely using self-play deep reinforcement learning that is a super-human level intelligent gameplaying agent.

Hearts game is also worked on in many machine learning studies. However, there are two main gaps in the state-of-the-art research in Hearts. First, none of the studies reached human-level performance. There are some studies achieving good results using the simplified versions of the game Hearts. Second, none of the studies presented convincing results against human players. In [17], Monte Carlo search-based methods were used, and the passing was excluded from the game. The trained agents learned Hearts; however, they were far from being an expert. In [18], decision rule algorithms were used. The agent was tested against commercial Hearts programs of that time, and it played on par with them. The agent developed in the study [18] was assumed as an expert player in another study [1]. The temporal difference learning method was used in this study with the primary concern of feature generation. Experimental results show that the developed agent beat its opponents, the agents from the study [18]. However, both agents were behind human players. Another study [19] also used temporal difference learning; however, it could not achieve good results. In a study by Ishii et al. [20], the researchers modeled the environment as a partially observable Markov decision process. They estimated cards of the opponents and predicted the action accordingly. They did not involve passing or shooting the moon in their study. However, they presented good results against rule-based agents. In [21], Hearts was learned by the combination of Monte Carlo learning and multilayer perceptrons, but researchers eventually declare that even though the resulting networks were able to beat some opponents the level of play was not near that of experts. In [22], researchers applied determinization to extend MCTS to imperfect information games by implementing it for a simplified version of the Hearts game (using 32 cards instead of 52 cards, i.e. a smaller search space) game. In a more recent work [23, 24], Zha et al., introduced a toolkit that can play and learn several card games. Their platform allows other people to extend it by adding new games and algorithms. They used Deep Q-Learning (DQN) [25], Neural Fictitious Self-Play (NFSP) [26]. Although introduced agents learn games to some promising degree, the work lacks a benchmark with human or strong rule-based agents. In another work, [23, 24] is extended with Deep Monte Carlo (DMC) using the game DouDizhu [27]. With the introduction of DMC, the proposed solution becomes the strongest DouDizhu AI program.

3. Hearts

Hearts is one of the most popular trick-taking card games around the world. Although there are many versions of the game[1], the most popular version is worked on for this study. Hearts is played with four players. It is played with a standard deck of 52 playing cards with four different suits (*clubs* ♣, *diamonds* ♢, *hearts* ♡, *spades* ♠) and 13 different ranks (*2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, *10*, *J*, *Q*, *K*, and *A* in increasing order). All suits have equal value (i.e. no suit has value over other suits). However, the values of the cards with the same suit increase by their ranks. Hearts is an imperfect information game because each player's cards are hidden from other players.

The game consists of rounds, and each round consists of 13 tricks. Each player is dealt 13 cards at the

[1]Wikipedia (2021). Hearts (card game) - Wikipedia [online]. Website https://en.wikipedia.org/wiki/Hearts_(card_game) [accessed 11 October 2021].

beginning of a round. The round ends when players play all their cards (i.e. when 13 tricks are over). At the end of each round, penalty points are calculated for each player. Those points are added to the scores of the players. The game ends when a player reaches or exceeds a preset number of points (e.g., 100). Hearts players aim to take as few points as possible (i.e. as few penalties as possible) at the end of the game.

Each Hearts round has two consecutive phases: the passing phase and the playing phase. In the passing phase, each player selects 3 of their cards to pass to the other players. Each player gives selected cards to the player sitting on their left in the first round, to the player sitting on their right in the second round, and to the player sitting across in the third round. In the fourth round, players do not select or pass any cards. Players give selected cards as face down, and they do not look at their new cards before all players give their selected cards. This passing behavior repeats for the following rounds until the game ends.

After the passing phase, the playing phase starts. In the playing phase, players play all 13 tricks by throwing their cards, one for each trick. There are some rules to follow for picking cards to throw. The player who holds *2 of clubs* (*2*♣) throws it to start the first trick. The first card of a trick is called the leading card, and its suit is called the leading suit. The rest of the cards of a trick are called the following cards. The player who picks a leading card can throw a card with any suit but hearts unless one of the players has previously thrown a *hearts card* (♡) in that round. Other players who pick the following cards must throw a card with the leading suit (i.e. they must follow the leading suit). If they have no such card, they can throw a card with any suit. The only exception is that players can not throw a *hearts card* (♡) or the *queen of spades* (*Q*♠) as the following card of a round's first trick.

The winner of a trick is determined according to the four thrown cards. The player who throws the highest-ranked card with the leading suit wins the trick, takes the cards, and starts the next trick of the round. Players are given cumulative penalty points at the end of each round according to the cards they take. Players are penalized 1 point for each *hearts card* (♡) they take and 13 points if they take the *queen of spades* (*Q*♠) card. So, 26 penalty points are distributed among the players for each round with an exceptional case in which taking penalized cards is not an undesired thing. Players who collect all 26 penalty points in a round receive 0 penalty points (instead of receiving 26 penalty points) while other players have penalized 26 points each. This exceptional case is called *shooting the moon.* The game ends when a player reaches or exceeds a preset number of points, and the player with the lowest penalty points wins the game.

4. Proposed work

Since Hearts includes two consecutive phases, the passing phase, and the playing phase, a Hearts playing intelligent agent should be capable of both (1) selecting cards to give to other players at the beginning of game rounds and (2) picking cards to throw during the tricks of game rounds. This two-phased gameplaying makes Hearts hard to play and, as a result, hard to develop intelligent agents for. To ease the development process, the agent developed in the scope of this study is divided into subagents such that each subagent is responsible for learning a part of the game. The Hearts playing intelligent agent uses all subagents to make decisions during the game. The usage of the subagents during a game round is depicted in the flow diagram in Figure 1.

So basically, three types of learning subagents were developed: agents responsible for playing (i.e. *playing agents*), agents responsible for passing (i.e. *passing agents*), and agents that decide the next play style (i.e. *decision agents*). However, a single passing agent and a single playing agent were not enough for Hearts because it is complex and includes different gameplaying strategies.

Playing agents: Hearts players have three main styles of playing according to their intention about taking penalized cards: playing not to take penalized cards, playing to take all penalized cards (i.e. shooting the moon), and playing not to allow other players to take all penalized cards (i.e. not allowing them shooting the moon). So, three playing agents were developed as subagents for all styles of playing. Those agents are called *"normal playing agent"*, *"moonshot playing agent"*, and *"moonshot breaker playing agent"*, respectively.

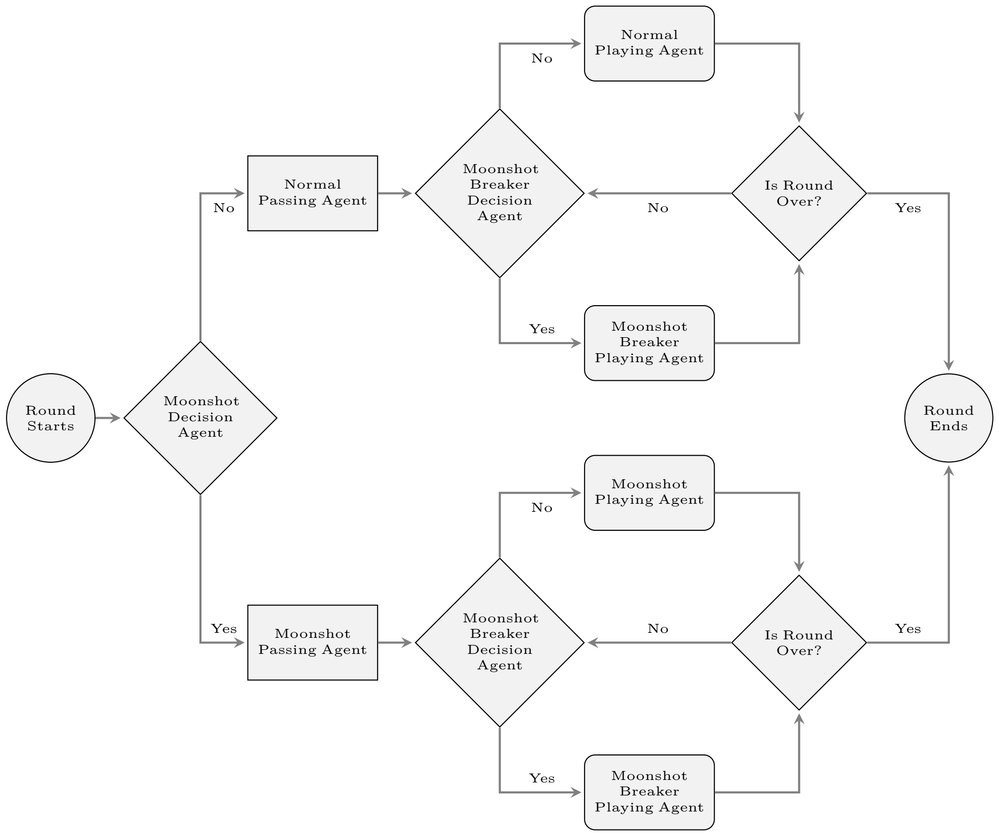

Figure 1. Flow diagram of all subagents that the Hearts playing agent uses to make decisions during a game.

Passing agents: Since the passing phase precedes and supports the playing phase, Hearts players also have different styles of passing to support different styles of playing: passing to support playing not to take penalized cards (i.e. passing to support normal playing), and passing to support playing to take all penalized cards (i.e. passing to support shooting the moon). So, two passing agents were developed as subagents for all styles of passing. Those agents are called *"normal passing agent"* and *"moonshot passing agent"*, respectively. Although there was another style of playing, which does not allow other players to take all penalized cards, no passing agent was developed to support that because in the passing phase information available to the players and the learning agents is limited. It is impossible to understand if some other player is going to try shooting the moon. So, it is only logical to not start the round with the intention of breaking moon shot attempts of other players. All five playing and passing agents are reinforcement learning agents.

Decision agents: As described in previous paragraphs, there are multiple styles of gameplaying and multiple corresponding subagents. Since the Hearts playing intelligent agent uses all subagents, a mechanism to decide which subagents (i.e. which styles of gameplaying) to use is needed. So, one more type of learning subagents was also developed: agents responsible for deciding the style of gameplaying (i.e. decision agents). Hearts players have two main decisions about the style of their gameplaying: the decision to play to shoot the moon or not, and the decision to play not to allow other players to shoot the moon or not. So, two decision agents were developed as subagents for all decisions. Those agents are called *"moonshot decision agent"* and *"moonshot breaker decision agent"*, respectively. Decision agents are supervised learning agents.

In this study, artificial neural networks were used for training both the RL agents and the SL agents. The neural networks used for RL agents take game states (i.e. states of the cards) as input and the rewards for corresponding game states as output. The neural networks used for SL agents also take game states (i.e. initial states of the distributed cards) as input and the correct decisions as output. All neural networks in this study are fully connected multilayer feedforward neural networks. Each network consists of an input layer, a hidden layer, and an output layer. All nodes in the layers of the neural networks use the sigmoid activation function. The architecture of the neural networks is depicted in Figure 2. The neural networks were trained by backpropagation algorithm using stochastic gradient descent (SGD) optimization strategy and mean squared error (MSE) loss function.

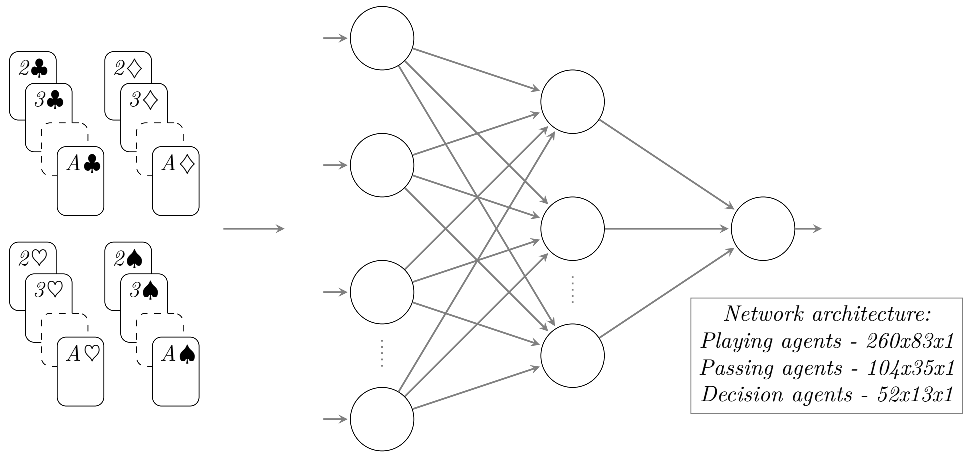
2♣
3♣
A♣
2♢
3♢
A♢
2♡
3♡
A♡
2♠
3♠
A♠
Network architecture:
Playing agents - 260x83x1
Passing agents - 104x35x1
Decision agents - 52x13x1

Figure 2. A simplified representation of the architecture of neural networks used for all subagents, i.e. playing agents, passing agents, and decision agents.

4.1. Game state representations

State representations have an essential role in the success of training learning agents. In this study, passing agents and playing agents use different sets of features and state representations because of their natures. However, they use the same representations within themselves. In other words, two passing agents have a shared state representation, and three playing agents have a different (from passing agents) but shared state representation. Two decision agents also use the same state representation with the playing agents. Since players cannot see the cards in other players' hands, Hearts is an imperfect information game. So, information about players in the game is limited to their cards and already thrown cards. With this limited information, *two player-centric state representations based on positions of the cards* were constructed for the passing and the playing agents (and indirectly for the decision agents).

In *the player-centric state representation for the playing agents*, a card can be in five different positions: in the hand of the player who is at the center of the state representation (*[1, 0, 0, 0, 0]*), in either of the other players' hands (*[0, 1, 0, 0, 0]*), on the table by being thrown for the current trick by the player (*[0, 0, 1, 0, 0]*), on the table by being thrown for the current trick by the other players (*[0, 0, 0, 1, 0]*), or off the table by being already thrown by a player in earlier tricks (*[0, 0, 0, 0, 1]*). Each of these five possible positions of a card is represented by a one-hot vector of length 5. The complete state is represented by 52 one-hot vectors of length 5. Each vector corresponds to a card from the standard 52 cards deck sorted by suit and rank in ascending order (from *2 of clubs* (*2*♣) to *ace of spades* (*A*♠)).

Some of the possible card positions for the playing agents are not available for the passing agents because the passing phase precedes the playing phase and occurs at the beginning of the game rounds. In *the player-centric state representation for the passing agents*, a card can be in just two different positions: in the hand of the player at the center of the state representation or in either of the other players' hands. This state representation also includes the information that if the card in the player's hand is selected for passing or not. So, there are two possibilities for a card in the player's hand: it is selected for passing or not selected for passing. The state representation for the passing agents is the concatenation of two vectors of length 52. Each bit in both vectors corresponds to a card from the standard 52 cards deck sorted in the same order in the state representation for the playing agents. In the first vector, each bit tells if the player has the card (*1*) or one of the other players has the card (*0*). In the second vector, each bit tells if the card is selected for passing (*1*) or not selected (*0*).

4.2. Reinforcement learning agents

All playing agents in this project (namely normal playing agent, moonshot playing agent, and moonshot breaker playing agent) are RL agents. They have similar structures with the same game state representation but different rewarding rules. In Hearts, each game round consists of 13 tricks. The playing agents are trained at the end of each round. Each action in a trick causes a change in the state. So, each player has 13 states when a round ends. Unlike the classical RL approach, where a discounted reward is applied to earlier states, each state is given the same terminal reward in this study due to the nature of Hearts and some other card games. All actions of a player (i.e. all 13 cards thrown by the player) are counted equally important, independent of their chronological order.

Terminal rewards given to different playing agents are given in Table 1, in which *round_penalty* is the number of penalty points the player receives at the end of a round. The purpose of the normal playing agent (i.e. the agent playing not to take penalized cards) is to take as few penalty points as possible, thus taking fewer penalty points is rewarded. If the agent takes more than 10 penalty points, it is considered that the agent fails in the round, and no reward is given. The upper limit 10 for penalty points was chosen with a mini-experiment. The straightforward approach to give a terminal reward for the normal playing agent is to give 0 when the agent takes all 26 penalty points and give 1 when the player takes no points. However, a player generally takes at most 13 penalty points in a normal game. So, training with an upper limit of 13 would converge better than training with an upper limit of 26. Furthermore, upper limits less than 13 would force the agent to receive fewer penalties and play better. In the mini-experiment to determine hyperparameter for the upper limit, we tested numbers 5 through 13. Since number 10 gave the best performance, it was chosen as the upper limit. The moonshot playing agent must take all penalty points to succeed, which means no mistake is allowed: if the agent takes all penalty points, the maximum reward is given; otherwise, no reward is given. Rewarding for the moonshot breaker playing agent is slightly more complicated than the others. Since the agent must prevent others from shooting the moon, succeeding in doing so is more prioritized than taking fewer penalty points. If one of the other players shoots the moon, the agent receives no reward. However, if no one shoots the moon, the agent receives some reward.

Table 1. Terminal rewards used for training three playing agents.

Normal playing agent
$terminal\_reward = max(0, \frac{10 - round\_penalty}{10})$
Moonshot playing agent
$terminal\_reward = \begin{cases} 1 & \text{if } round\_penalty = 26 \\ 0 & \text{otherwise} \end{cases}$
Moonshot breaker playing agent
$terminal\_reward = \begin{cases} 1 & \text{no one shoots the moon and the agent takes no penalty points} \\ 0.1 & \text{no one shoots the moon and the agent takes some penalty points} \\ 0 & \text{otherwise} \end{cases}$

Besides giving each state the same reward as a terminal reward, *immediate rewards* are also given to each state so that the agents can distinguish good and bad actions in a game round. Immediate rewards serve as improvements rather than driving the learning. Immediate rewards given to different playing agents are listed in Table 2, in which *trick_penalty* is the number of penalty points the player receives at the end of a trick. The immediate rewards are limited in the range $[0, 0.5]$ because they must not shadow terminal rewards. The normal playing agent is given some positive reward if it takes no penalty points in a trick to encourage not taking penalty points. If the agent takes more than 4 penalty points, no reward is given. That upper limit for penalty points for immediate rewards was decided similarly to the upper limit for penalty points for terminal rewards. We have observed that a player generally takes at most 4 points in a trick. Then the numbers 1 through 4 were tested as the upper limit. Since number 4 gave the best performance, it was chosen as the upper limit. On the contrary, the moonshot playing agent is encouraged to take penalty points. For the moonshot breaker playing agent, a positive immediate reward is given only to one state (i.e. one trick) in which the possibility of someone shooting the moon just disappears. That state occurs when a player wins a trick in which penalized cards exist, takes penalty points, and becomes the second player who takes penalty points in a game round. If the agent becomes the second player in such a state, it receives a partial reward; otherwise, it receives the full reward.

Both passing agents in this study (namely normal passing agent and moonshot passing agent) are also RL agents. Like playing agents, passing agents also have similar structures, same state representation but different rewarding rules. Recognizing the success of a passing agent by looking only at the passing phase is impossible. A player's success at passing is determined at the end of the playing phase, and it depends on the player's success at playing. For example, a moonshot passing agent who selects cards to pass to shoot the moon is counted as successful at the passing phase if he achieves shooting the moon at the playing phase. That is why the passing agents are trained with the help of the pretrained playing agent counterparts. So, the passing agents are also trained at the end of each round. Each player has a single state when a round ends, representing the player's initial cards and the cards selected for passing, and it is given the terminal reward. Terminal rewards given to different passing agents are the same as their playing agent counterparts. Besides terminal rewards, no immediate rewards are given to passing agents because there is no intermediate state for passing agents that can affect the passing performance.

Table 2. Immediate rewards used for training three playing agents.

Normal playing agent
$immediate\_reward = max(0, \frac{4 - trick\_penalty}{8})$
Moonshot playing agent
$immediate\_reward = min(\frac{trick\_penalty}{10}, 0.5)$
Moonshot breaker playing agent
$immediate\_reward = \begin{cases} 1 & \text{a second player takes penalty points and it is not the agent} \\ 0.1 & \text{a second player takes penalty points and it is the agent} \end{cases}$

After training is done for playing and passing agents, one final training is done to increase the success of both agents where they are concurrently trained. We refer to that final training as the *fine-tuning* process. Two phases of the Hearts game, passing and playing, are equally important and inseparable when deciding on a strategy. In other words, passing cards well and playing cards well are important on their own, yet their cooperation is crucial. One must include the information of the passed cards to decide the next card to play in the playing phase and vice versa. During the training of the passing agents, pretrained playing agents continue to play the game after the passing phase. It means that the passing agents learn passing while teaming up with the playing agents. Therefore, the passing agents learn to utilize the overall performance of this little system containing the passing agents and the pretrained playing agents. However, the pretrained playing agents are not aware of the passing phase at all. Their performance can be enhanced if passed card information is fed to the agents. To solve this problem, fine-tuning is applied. During the fine-tuning process, both passing and playing agents are trained and parameters of their networks are updated in the same epochs. This final step of training dramatically increased the success of the passing and playing agents because they adjusted their decisions concerning each other.

4.3. Supervised learning agents

Both of the decision agents in this study (namely moonshot decision agent and moonshot breaker decision agent) are SL agents. They were trained on labeled data, which was generated using the pretrained passing and playing agents. Each agent's dataset is separate, and both datasets contain one million samples. Each sample in datasets is a state-label pair in which the state represents a player's cards, as previously described in this section. Each sample is labeled as one if the corresponding passing and playing agents succeed in shooting the moon; otherwise, it is labeled as zero.

The decision agents have two critical roles: the first is to decide which RL agents to use, and the second is to participate in the training of RL agents that are counterparts of the decision agents. When training RL agents, one challenge is that the games they play are highly unbalanced in terms of moonshot count. On average, only 5% of games result in a moonshot. To overcome that challenge, the decision agents are used to filter out some games before training RL agents to train on more balanced data. Thus the overall training process is repeated that way.

5. Experiments

Throughout the experiments in this study, trained agents were compared against three different opponent Hearts players. The first one is a *Monte Carlo Tree Search (MCTS) player*, which is used in the mobile application with the name *"Hearts - Free Card Games"* (name may subject to change) by *SNG Games*[2] (a Turkey-based mobile game development company). That MCTS player is based on the study by Santo and Wong [17]. However, although the player in that study does not implement passing and is not good at shooting the moon, the MCTS player used in this study includes some rule-based implementation for passing and shooting the moon. The second opponent is a *Deep Q-Network (DQN) agent* which is implemented by extending the RLCard toolkit provided by [24]. In the toolkit Hearts game is not implemented, however, due to the lack of intelligent agents to test our work, we extended the toolkit with Hearts. In this version of Hearts game, moon shot is included yet passing is left out since the toolkit does not support such a game phase. DQN agent is trained using the same state, action space and rewards as our trained agents. It is trained for 500 thousand games until there is no significant improvement against random agents. The third opponent of the trained agents is the *human players*, who are the users of the mentioned mobile application. Trained agents in this study are integrated into that mobile Hearts game, and they are tested against human players from all over the world.

The success of the agents is measured with two metrics: *winning ratio* and *average points*. The winning ratio is the percentage of games the agent wins (i.e.the percentage of games in which the agent takes the lowest penalty points among others at the end of the game) in test games. Winning ratios in the tables show the averaged values, which means the winning ratio of three of the same type of agents is divided by three. For example, when three trained agents were compared against a single opponent, the winning ratios of trained agents are summed and divided by three, and the winning ratio of the single opponent is left as it is. The average points are the average number of penalty points the agent takes per round in the test games. In order to measure the success of the moonshot related agents, one additional metric is used: *moonshot success ratio*. The moonshot success ratio is the percentage of successful moonshot attempts by the agent.

In experiments, either single trained agents were compared against three opponents or three trained agents were compared against a single opponent. Thus, a winning ratio value of 25% means that the trained agent plays equally well against its opponents, and a bigger value means that it plays better than its opponents. Moreover, since there are four players and 26 penalty points to share in a game round, an average points value below 6.5 tells that the trained agent plays better than its opponents. Since many separate agents were trained in this study, their success was evaluated both individually and collectively. For each experiment carried out in this study to test a trained agent's performance individually or collectively, 1000 separate test games were played between the tested agent and its opponent agents.

As the first experiment, the success of MCTS player was measured against human players. The purpose of this experiment was to compare the performances of the two opponents of our trained agents because their performances will be a reference to measure the performances of our trained agents. In the experiment, one human player is played against three MCTS players. The human players in this experiment are also the players of the mobile game mentioned before. The results of this experiment show that the winning ratio of humans is close to 50% whereas a single MCTS player wins 17% of the games on average.

[2]SNG Games (2021). Hearts - Free Card Games [online]. Website https://play.google.com/store/apps/details?id=com.sngict.Hearts&hl=en [accessed 11 October 2021].

5.1. Normal gameplaying style agents

Table 3 contains all of the statistics of normal gameplaying style agents against opponent agents on various conditions. The first row of the table contains the statistics about the games in which the normal playing agent is tested against three MCTS players. During these tests, *passing and moonshot rules of the Hearts game were disabled* so that the normal playing agent's success can be measured correctly. The average points and the winning ratio values of both the trained player and the opponent players are presented in the table. The presented values reveal that the normal playing agent achieves to beat MCTS players in terms of both the winning ratio and the average points.

Table 3. Statistics about the test games between trained agents (normal playing and normal passing) against various opponent agents.

		Winning ratio	Average points
A trained player (uses normal playing agent) and three MCTS players	Trained player	26.60%	6.32
	MCTS players	24.47%	6.56
A trained player (uses normal playing agent and normal passing agent (before fine-tuning)) and three MCTS players	Trained player	26.30%	6.32
	MCTS players	24.57%	6.56
A trained player (uses normal playing agent and normal passing agent (after fine-tuning)) and three MCTS players	Trained player	30.30%	5.93
	MCTS players	23.23%	6.69
A trained player (uses normal playing agent) and three DQN agents	Trained player	28.00%	-
	DQN players	24.00%	-
Three trained players (uses normal playing agents and normal passing agents (after fine-tuning)) and a human player	Trained player	26.10%	6.41
	Human player	21.70%	6.76

The second row of the Table 3 presents the normal playing agent and the normal passing agent's collective success against three MCTS players before the playing and the passing agents are fine-tuned. During the tests, *the passing rule was introduced again*; however, *the moonshot rule was still disabled*. In the test games, the passing phase is played by the normal passing agent, and the normal playing agent plays the rest. The values presented in the second row of Table 3 shows that even without a fine-tuning process, the trained player performs better than the MCTS players. However, the normal passing agent's introduction does not remarkably change the winning ratio or the average points.

In the third row of Table 3, the normal playing agent and the normal passing agent's collective success against three MCTS players after the two agents are fine-tuned are presented. These agents (i.e. subagents) are the finalized versions of the normal gameplaying style agents. As shown in the third row of Table 3, fine-tuning of the corresponding playing and passing agents dramatically increases their collective success.

Results against the DQN agents after fine-tuning are presented in the fourth row of Table 3. Since the RLCard toolkit does not support the passing phase, only a normal gameplaying agent is played against the DQN. To have a fair comparison, the DQN agent in this experiment is trained only to learn normal gameplay without passing or moon shot. With the explained setup, the trained agent outperforms the DQN agent by winning 28% of the games.

The finalized versions of the normal gameplaying style agents were also compared against previously mentioned human players. The fifth row of the Table 3 presents collective success of the finalized normal gameplaying style agents against human players. Unlike the comparisons against MCTS players, three trained agents are compared against single human players in comparisons against human players. The fifth row of the Table 3 shows that the average value of the winning ratio of the trained players is higher than 25%, and their average points value is lower than the human player's average points value. Hence, it can be concluded that *the trained players are better than human players without the moonshot rule of the game.*

5.2. Moonshot gameplaying style agents

Three subagents were developed for the moonshot gameplaying style: moonshot passing agent, moonshot playing agent, and moonshot decision agent. As stated in Section 4, the moonshot decision agent is a supervised learning agent, and it was trained on labeled data generated using the pretrained moonshot passing and moonshot playing agents. The decision agent was trained using 80% of the data, and its performance was tested using the remaining 20%. After the training, the loss of the trained network was measured as 10%. The moonshot decision agent outputs a probability value, which is the player's likelihood of shooting the moon. If the probability is greater than 96%, the moonshot gameplaying style agents play the game round; otherwise, the normal gameplaying style agents play. The threshold value of 96 was decided after some additional experiments.

The experiments in this section are an extension to the experiments in Section 5.1 by *introducing the moonshot rule back* and *adding the moonshot gameplaying style agents to the normal ones.* Hence, in these experiments, the performance of the moonshot gameplaying style agents and their contribution to the Hearts playing intelligent agent were measured.

Table 4 contains all of the statistics of all normal and moonshot agents against opponent agents under various conditions. The first row of Table 4 presents the collective success of the moonshot decision agent, the moonshot playing agent, and the normal playing agents against three MCTS players. Comparing the values in the first row of the Table 4 with the values in the first row of the Table 3 for which *the passing and moonshot rules were disabled* and *the moonshot gameplaying style agents were not available* during the tests, the winning ratio and the average points values of the trained agents in both experiments are very close. In both cases, the trained players achieve to beat MCTS players. There are a couple of important things worth mentioning here. First of all, although MCTS players are not quite good at shooting the moon, they are good at not allowing other players to shoot the moon (i.e. breaking the moonshot). This capability of the MCTS players decreases the success of their opponents against them. In this experiment, the trained player's moonshot success ratio, which is the percentage of successful moonshot attempts by the trained player, is 21.4% against MCTS players. Another reason for the low moonshot success ratio is that the moonshot decision agent used in this experiment is an early version, which does not perform as well as the finalized version. The finalized moonshot decision agent was trained by both the moonshot passing agent and the moonshot playing agent. However, this early version used in this experiment was trained by only the moonshot playing agent because the moonshot passing agent was not trained yet.

In the second row of Table 4, the collective success of the moonshot passing agent, the moonshot playing agent, the finalized version of the moonshot decision agent, and all normal gameplaying style agents against three MCTS players are presented. Comparing the values in the second row of the Table 4 with the values in the second row of the Table 3 for which *only the moonshot rule was disabled* and *the moonshot gameplaying style agents were not available* during the tests, it is obvious that the trained player in this experiment is more successful thanks to its moonshot gameplaying style capabilities. The trained player in this experiment is also slightly better than the trained player in the experiment presented in the third row of the Table 3, which uses the fine-tuned versions of the normal playing agent and the normal passing agent. Moreover, the trained player's moonshot success ratio against MCTS players is increased to 52.24%.

The third row of Table 4 presents the comparison between the finalized versions of moonshot gameplaying style agents in collaboration with normal gameplaying style agents against the DQN agents. The table shows us that the trained agents are winning a bigger percentage of the games as well. We can also compare the results against the fourth row of the Table 3 and see that adding the moon shot increases the winning percentage of the trained agents which means they are better at both shooting the moon and normal gameplay.

The fourth row of Table 4 presents the comparison of the finalized versions of all moonshot gameplaying style agents in collaboration with all normal gameplaying style agents against human players. This experiment is an extension to the experiment presented in the fifth row of Table 3 by *introducing the moonshot rule back* and *making the moonshot gameplaying style agents available*. By comparing the values in the fourth row of Table 4 and the fifth row of Table 3, it is observable that the average winning ratio of the trained players is decreased even though the average points of human players and trained agents are close. These values tell that human players are better at shooting the moon than trained agents. Moreover, the trained agent's moonshot success ratio is 30%. So, it is evident that the human players managed to prevent trained agents from shooting the moon in most cases. There is one more important statistic to consider. In the experiment whose results were presented in the fifth row of the Table 3, the moonshot rule was disabled and no moonshot gameplaying style agents were available for the trained players. If we introduce the moonshot rule back (but still do not make moonshot gameplaying style agents available) and repeat the experiment, the winning ratio of the trained players becomes 19%. So, comparing the value 19% with the winning ratio of the trained players in this experiment, it can be concluded that the introduction of moonshot gameplaying style agents increases the performance of the trained players but not enough to strictly beat human players. So, *the trained players are better than non-human players and they can compete with human players.*

Table 4. Statistics about the test games between trained agents (normal playing, normal passing, moonshot playing, moonshot passing and moonshot decision) against various opponent agents.

		Winning ratio	Average points
A trained player (uses moonshot decision agent (early version), moonshot playing agent and normal playing agent) and three MCTS players	Trained player	26.10%	6.74
	MCTS players	24.63%	7.63
A trained player (uses moonshot decision agent (finalized version), moonshot passing agent, moonshot playing agent and all normal gameplaying style agents) and three MCTS players	Trained player	32.00%	6.27
	MCTS players	22.67%	7.71
A trained player (uses moonshot and normal gameplaying style agents) and three DQN agents	Trained player	32.00%	-
	DQN players	22.67%	-
Three trained players (use all moonshot gameplaying style agents and all normal gameplaying style agents) and a human player	Trained player	23.33%	7.65
	Human player	30.00%	7.57

5.3. Moonshot breaker gameplaying style agents

Two subagents were developed for the moonshot breaker gameplaying style: moonshot breaker playing agent and moonshot breaker decision agent. Experiments in this section are an extension to the experiments in Section 5.2 by *adding the moonshot breaker gameplaying style agents to all agents so far*. Hence, in these experiments, the performance of the moonshot breaker gameplaying style agents and their contribution to the Hearts playing intelligent agent were measured. Like the moonshot decision agent, the moonshot breaker decision agent is also a supervised learning agent, and it was trained on labeled data generated using the pretrained moonshot breaker playing agents. 20% of the data was reserved for testing the decision agent's performance, and the rest was used for training the decision agent. After the training, the loss of the trained network was measured as 10%.

Since MCTS players are not good at shooting the moon, the moonshot breaker gameplaying style agents are tested against (1) the trained players with no moonshot breaking capability (i.e. players use all moonshot and normal gameplaying style agents) and (2) human players. Table 5 presents statistics about the test games between trained players. The first row is for the test games between one trained player with moonshot breaking capability and three trained agents without moonshot breaking capability. The second row is for the test games between four trained agents without moonshot breaking capability. Statistics in the second row form the baseline to measure the success of the moonshot breaker gameplaying style agents which is presented in the first row. The values presented in Table 5 reveal that adding moonshot breaking capability resulted in fewer winnings for the trained player. However, that trained player achieved to cut down the moonshot count of its opponents almost in half by decreasing it from 111 to 61.

The performance of the trained players with moonshot breaking capability (i.e. the trained players that use all subagents) against human players is presented in Table 6. Compared to the winning ratio of 23.33% in the fourth row of the Table 4; the winning ratio decreased to 22.67% with the introduction of moonshot breaker gameplaying style agents. Moreover, the average penalty points that the trained players receive increased by about 2%. However, comparing the moonshot success ratios of the human players in the experiments presented in the fourth row of Table 4 and Table 6, the introduction of moonshot breaker gameplaying agents causes about 10% loss. On the other hand, the moonshot success ratio of the opponents of a trained player against other trained players without moonshot breaking capability decreases almost to half (i.e. about 50% loss) with the introduction of moonshot breaker gameplaying style agents, as presented in Table 5. The reason behind this difference is that the moonshot breaker decision agent was not trained on a dataset collected from human players; it was collected from the trained agents. To achieve more success against human players, the decision agent needs to know how humans play, and so it needs to be trained by human players. So, *it is possible to increase the success of the moonshot breaker gameplaying style agents with better datasets (i.e. teachers).*

Table 5. Statistics about the test games between four trained players with moonshot breaking capability is enabled for one of them or not.

Moonshot breaking capability	Winning ratio	Average points	Total moonshot count of opponents
Enabled	22.50%	7.38	61
Disabled	24.80%	7.48	111

Table 6. Statistics about the test games between three trained players (use all moonshot breaker, moonshot, and normal gameplaying style agents) and a human player.

	Winning ratio	Average points
Trained agent	22.67%	7.81
Human player	32.00%	7.22

6. Conclusion

In this study, a self-learning intelligent system was developed for the Hearts card game which has some challenging properties. Hearts has a big state space with various rules and phases like passing and shooting the moon. Prior works tackle the problem with a single intelligent agent or by removing important rules of Hearts. Unlike prior work, we introduced a collaborative collection of subagents to solve the problem.

The two main goals of the study are to develop a system that can play Hearts on a human level and present the results by playing against a huge number of human players. Experimental results show that the proposed system can beat MCTS agents (32% win rate), DQN agents (32% win rate), and even with a complete version of Hearts game is learned to compete with humans (22.67% win rate) which makes it the strongest intelligent system for Hearts so far. Moreover testing the proposed system against humans can be a baseline for future work.

The decentralized way of training used in this study enables each subagent to focus on relatively simple problems. Thanks to this simplicity, we solved the problem of learning an imperfect information game with simple agents. In our specific study, cooperation issues between the subagents were solved by fine-tuning subagents to adjust corresponding passing and playing subagents to play more cooperatively and by introducing supervised subagents to select the next subagents to play.

References

[1] Sturtevant NR, White AM. Feature construction for reinforcement learning in hearts. In: International Conference on Computers and Games; 2006. pp. 122–134.

[2] Alzubi OA. A deep learning-based frechet and dirichlet model for intrusion detection in IWSN. Journal of Intelligent & Fuzzy Systems 2021; (Preprint): 1-11.

[3] Samuel AL. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development 1959; 3 (3): 210-229.

[4] Samuel AL. Some studies in machine learning using the game of checkers. ii—recent progress. IBM Journal of Research and Development 1967; 11 (6): 601-617.

[5] Tesauro G. Temporal difference learning and TD-Gammon. Communications of the ACM 1995; 38 (3): 58-68.

[6] Tesauro G. Programming backgammon using self-teaching neural nets. Artificial Intelligence 2002; 134 (1-2): 181-199.

[7] Sutton RS. Learning to predict by the methods of temporal differences. Machine Learning 1988; 3 (1): 9-44.

[8] Campbell M, Hoane Jr AJ, Hsu F. Deep Blue. Artificial Intelligence 2002; 134 (1-2): 57-83.

[9] Hsu F. Behind Deep Blue: Building the computer that defeated the world chess champion. Princeton, NJ, USA: Princeton University Press, 2002.

[10] Silver D, Huang A, Maddison CJ, Guez A, Sifre L et al. Mastering the game of go with deep neural networks and tree search. Nature 2016; 529 (7587): 484-489.

[11] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A et al. Mastering the game of go without human knowledge. Nature 2017; 550 (7676): 354-359.

[12] Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 2018; 362 (6419): 1140-1144.

[13] Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L et al. Mastering atari, go, chess and shogi by planning with a learned model. Nature 2020; 588 (7839): 604-609.

[14] Niklaus J, Alberti M, Pondenkandath V, Ingold R, Liwicki M. Survey of artificial intelligence for card games and its application to the swiss game jass. In: 2019 6th Swiss Conference on Data Science (SDS); Bern, Switzerland; 2019. pp. 25-30.

[15] Baykal O, Alpaslan FN. Reinforcement learning in card game environments using monte carlo methods and artificial neural networks. In: 2019 4th International Conference on Computer Science and Engineering (UBMK); 2019. pp. 1-6.

[16] Charlesworth H. Application of self-play reinforcement learning to a four-player game of imperfect information. ArXiv 2018; abs/1808.10442.

[17] Santo WL, Wong A. Evaluating mcts in a new ai framework for hearts. Retrieved October 11, 2021, from https://wellssanto.com/HeartsAI.pdf

[18] Sturtevant NR. Multi-player games: algorithms and approaches. PhD, University of California, Los Angeles, CA, USA, 2003.

[19] Kuvayev L. Learning to play Hearts. In: AAAI/IAAI; 1997. pp. 836.

[20] Ishii S, Fujita H, Mitsutake M, Yamazaki T, Matsuda J, Matsuno Y. A reinforcement learning scheme for a partially-observable multi-agent game. Machine Learning 2005; 59 (1): 31-54.

[21] Wagenaar M. Learning to play the game of hearts using reinforcement learning and a multi-layer perceptron. BS, University of Groningen, Groningen, The Netherlands, 2017.

[22] Bax F. Determinization with Monte Carlo Tree Search for the card game Hearts. BS, University of Utrecht, Utrecht, The Netherlands, 2020.

[23] Zha D, Lai KH, Cao Y, Huang S, Wei R et al. RLCard: a toolkit for reinforcement learning in card games. ArXiv 2019; abs/1910.04376.

[24] Zha D, Lai KH, Huang S, Cao Y, Reddy K et al. RLCard: a platform for reinforcement learning in card games. In: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence; 2021. pp. 5264-5266.

[25] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J et al. Human-level control through deep reinforcement learning. Nature 2015; 518 (7540): 529-533.

[26] Heinrich J, Silver D. Deep reinforcement learning from self-play in imperfect-information games. arXiv preprint arXiv:1603.01121.

[27] Zha D, Xie J, Ma W, Zhang S, Lian X et al. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In: International Conference on Machine Learning; 2021. pp. 12333-12344.