Report:
On cooperative multi-agent planning with incomplete information in games
Team 59

Cybill Clerger
cybill@kth.se

Fabien Lenthy
fabienlt@kth.se

Quentin Leroy
qleroy@kth.se

Bastien Ponchon
ponchon@kth.se

October 14, 2016

Abstract

A famous application of artificial intelligence is games, where it is commonly used to implement human-like behaving bots. In order to achieve this goal, one usually has to rely on planning where the artificial agent will have to plan a sequence of actions depending on a given state and leading to given goal. In this report we will deal with such planning in a very specific cooperative game with incomplete information, in which different agents will communicate and act in order to achieve a common goal. After exposing the formalism we adopted to represent and implement (simplified) rules and states of the game, we chose methods and heuristics to lead our agents to the desired goals in our Java application. Even though we never managed to reach the maximum score of the game, our agents performed quite well on average both with and without relaxed rules.

Contents

1 Introduction to the project topic

The overall subject of our project is planning in a cooperative game with incomplete information. In this situation, several agents have to work together in order to achieve a common goal. This implies that each agent will in turn try to infer the sequence of actions of the following agents depending on the state of the game and on the behaviour of the previously playing agents.

2 Inspiration: Collaborative Game Theory

Coming across a topic for our subject was not an easy task. The four of us being game theory enthusiasts, and board games being well suited field for studying planning, we quickly agreed on implementing one of those. The next day, we happened to be playing a collaborative game, where all players have to reach a common goal while having different incomplete information about the game state. Our subject had just been found.

We then found some papers and articles on the subject: Introduction to Planning in Multiagent Systems by Mathijs de Weerdt [1], Delft University of Technology, and Brad Clementy, Jet propulsion Laboratory, Planning algorithms by Steven M. LaValle [2] and Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information by Hirotaka Osawa [3]. These resources gave us, together with the course material, a better insight on how to tackle the subject.

3 Case study: Hanabi

3.1 Rules of the game

Hanabi is a card game that can be played from 2 to 5 players (there are 4 players in our setting). Those players need to cooperate to achieve a common goal, put on a fireworks show, which is in practice create stacks of cards of different colors in the right order. The catch is that the players cannot see their own cards, but only those of the other players (everybody holds out their cards face-out). That is where the collaboration comes into place: the players need to give each other clues, and ideally the best possible clues.

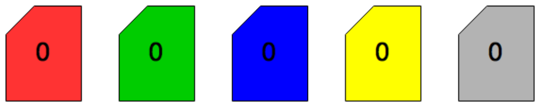

Figure 1: The initial state: the stacks of cards on the table are empty

Let's get into more details. The deck is constituted of cards having a color (red, green, blue, yellow, white) and a value (1, 2, 3, 4, 5). More precisely, there are in each color three 1s, two 2s, two 3s, two 4s and only one 5. Each player is dealt 4 cards in the beginning of the game (and will constantly have 4 cards in their hands, except maybe towards the end of the game, but we will come back to that later). The game also starts with 8 blue tokens representing the number of available hints and 3 red tokens representing the number of mistakes that are allowed.

At each turn, a player can perform one of three actions:

Give a hint If there is at least one blue token left, the player can spend it to give a hint to another player. The hints can be of two kinds. The *value* hints tells a player about all cards in his hand having that value. The *color* hints does the same but by telling about all the cards that are of this color. Note that we cannot tell a player just about the interesting cards, which makes clue giving challenging in a lot of situations.

Discard a card If we want to get one more blue token, we have to discard a card. If possible, we want to know that this card will not help us reach our goal, and we need to pay attention to the cards that appear only once in the remaining cards. After that, the player needs to draw (if possible).

Play a card That is what makes the game move on in a more concrete fashion. A player can choose to play one of their card if they are confident enough in the fact that it will extend a stack, i.e. if it starts a new color or if it has the value immediately following the one of the stack. Note that the player does not need to choose the stack they want to play on, it only matters if the card is playable. Fully completing a stack (i.e. playing a 5) gets us one more blue token. However, play a non-playable card removes a red token, and the card is discarded.

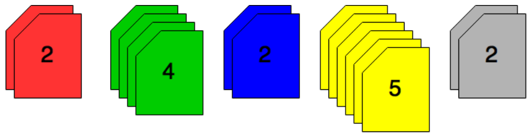

Figure 2: An intermediate state: the stacks are growing

Winning and losing are quite subjective things in that game, but anyway, there are several ways the game can end:

Perfect win Finish the game with a score of 25, i.e. fully completing every stack. This is sometimes hard to achieve, and possibly impossible for certain distributions. This actually is what we aim for when we play, because we're awesome perfectionists.

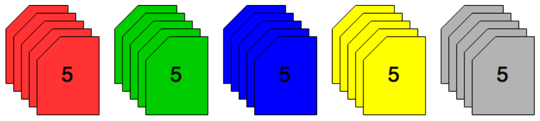

Figure 3: All the stacks are full: it is the best score possible

Complete loss Lose all three red tokens, i.e. make three mistakes. This is universally known as a loss. No way around it.

Natural end When a player draws the last card of the deck, each player gets to play exactly one turn and the game ends. The final score is the number of played cards. In the official rules, any score above 16 is considered as *excellent* and a 25 is *legendary*.

4 Approaches and methods

4.1 Multi-agent planning

Based on our reading of [1] we have thought about how to define precisely our problem in terms of a multi-agent planning problem, and what difficulties we will encounter when it comes to implementation. The general different phases of a multi-agent planning problem are as listed below:

- Assign goals to players
- Divide goals into smaller tasks
- Schedule the small tasks
- Communicate planning choices to other players
- Execute the plans

In our problem the agents are the players of the game. They play in turns and they all have a common goal which is basically to put a maximum number of cards on the table. The need for collaboration is of highest importance since the players cannot reach the goal alone, it is required that players take into account other players' actions in their plans as well as communicate with each other. There are many ways to see what a multi-agent planning problem is. Three axes are important when it comes to consider such a problem:

- Dependence of the actions and goals of each agent with regards to each other
- Cooperative or self-interested agents
- Quality of communication between the agents

Dependence of actions and goals Our problem can be defined as a multi-agent planning problem where actions and goals of each player are strongly related. Indeed each action performed by a player modifies the shared resources and there is a hierarchy in goals, that is to say that some goals cannot be reached before intermediate goals are achieved. The order of completion of the goals is actually conditioned by the rules of the games. Let us define the hierarchy of goals made possible by the rules of the game we have chosen. Let S^t be $\{S^t_R, S^t_G, S^t_B, S^t_Y, S^t_W\}$ where S^t_C defines the number of cards in the color C stack. At each turn the player can play a card and if it is a success it means that we are moving from one goal to another. In order to fully win the game the players will need to find a path in the tree of possible sequences of goals that lead the a state where all the stacks are full. This path would consist in 25 steps since the initial state is $\{0, 0, 0, 0, 0\}$.

Cooperation and shared resources The problem we tackle is fully cooperative. The agents are not self-interested, they all want to maximize the same utility which is basically the score of the current state game. In games there are often resources (in terms of information) available to all players, which will be referred to as shared resources, and resources that are specific to each player. For example in card games the played cards on the table faced up are known to all players whereas each player has his own card that only he can see. That is the most common situation in non-cooperative card games. But in our problem it is the other way around. Each player shows his cards to all other players but cannot see them himself. The cards in the hand of each player are not part of the shared resources, but they can be part of those thanks to certain actions that allow hints. In all cases after each turn the amount of shared resources are increased. Let $C^t = \{C_1, \ldots, C_m\}$ be the set of shared resources at turn t. A resource is a card which we know the value or color and

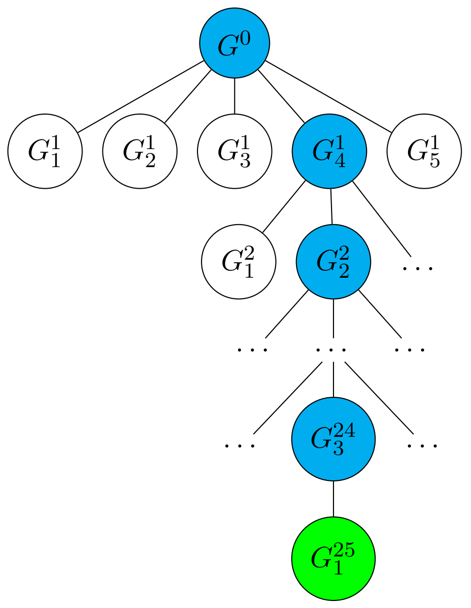

Figure 4: Hierarchy of goals: In blue is pictured a possible path that can lead to the ultimate goal

where it lies (on the table, in one players hand or in the graveyard). In our case, as the agents need to know something about their cards to be able to play them, it is in their best interest to share a maximum of information. In other words, we want the set of shared resources to be as large as possible. After a turn the size of the shared resources is increased, $|C^{t+1}| \geq |C^t|$. This can happen because a hint had been given or because a card has been played/discarded and became visible to every agent. In particular, when choosing a hint to give, the agents might want to give the one that maximizes the increase in shared resources.

Communication between the agents In order to optimally increase the amount of shared resources and therefore make it possible to reach intermediate goals, the agents should take advantage of their possibility to communicate between each other. In a game the communication between the players is restrained by the rules of the game. And in our cooperative problem the communication is always from one agent to every other agents. The agents cannot communicate directly their plan, they can only give hints on one players card(s). At a given state there are five possible intermediate goals that the players can immediately reached (unless towards the end of the game once 5s have been played, where there are less stacks to complete). Let us denote by $\hat{G}^t$ the last goal that was reached at turn t and $G^t = \{G^t_1, G^t_2, \dots, G^t_k\}$ the number of intermediate goals that can be reached at turn t, and $\widetilde{G^t_{P_i}}$ the goal that player k is planning to reach at turn t. Ideally $\widetilde{G^t_{P_1}} = \widetilde{G^t_{P_2}} = \widetilde{G^t_{P_3}} = \widetilde{G^t_{P_4}}$, but the rules of game forbid the agents to come to this sort of agreement. Given that the shared resources and the individual resources are different to all players, the players might not plan to reach the same goal. We have a problem of coordination. The players may try to reach two different possible intermediate goals whereas it would be more optimal that they all agree on the intermediate goal to reach in a minimum number of actions. However, since the agents do not plan in a distributed fashion but rather sequentially, this problem of coordination would be quite reduced in practice. Since the agents know the actions of previous players, they can adapt their plan. That way we can avoid most of the collisions we would have encountered in the case of agents acting in a parallel fashion.

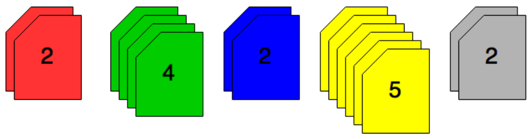

Figure 5: Example of intermediate goal

4.2 Certain states or uncertain states

The outcome of certain actions a player can take may be uncertain since the player does not have access to the complete information of the current state of the game. Hence a player can take an action which they do not know the consequence of, and it results in an uncertain state.

For instance, if an agent plays a card it does not have enough information on, they do not know if the card will be actually played or is it will be discarded. This makes simulating next actions and creating a plan even more challenging.

4.3 Information spaces

To better understand what to do with those uncertain states, we studied up on the notions of sensors and information spaces mainly using [2] When the environment is totally observable for the agent, we can create a plan by completely knowing the initial state and have no uncertainty about the effect of the different actions. For instance, a robot only gets information through its sensors and need to be able to plan ahead relying on the information gained by these sensors, and that even it is incomplete (or at least not available at each instant). In our case, the agent does not have access to all the information it needs: not only does it not know the order of the cards still in the deck but most importantly, it does have any knowledge on its cards if it did not receive any clues on them. This makes impossible the *classical* planning approach that relies on knowledge of the states. That is where information states come into play. Instead of expecting a perfect knowledge of the states, we want to create a plan only based on the actual information the agent has. One way the notion of information state can intuitively be interpreted is as a set of all the possible states the world can be in and still be consistent with the available information. Unfortunately, that approach is neither efficient or realistic in our case. When real people play Hanabi, they don't think of every possible hand. They prefer to actually use the information available to them and thus work around as much uncertainty as they can. The method we will use is close to that *block out the uncertainty* approach: when an agent hits an uncertain state during the conception of the plan, we choose to stop the search and we evaluate this state quite negatively. In other words, an agent will take only take an action with an uncertain outcome if it has no other choice. This allows our agents to come up with reasonable plans with regards to their limited information.

4.4 Forward search

In order to make a decision, an agent simulates the future actions of the following agents. An heuristic function is used to evaluate how good the state resulting from these actions is. However, as we lack information, information will be lost as we go deeper in the simulation. In our case, as a player can't see their own cards, the simulated next player will not be able to see their cards or the previous player's cards. So the fourth player in the simulation will not see any card, he will have to choose between playing or discarding a card. Simulating any further is pointless, as simulating a player who doesn't have any information and who couldn't receive any new hint is very likely to be like simulating a suicide turn where the player just discard an unknown card. Hence the depth of our forward search is limited by the number of players.

5 Implementation

5.1 Relaxed Rules

We chose to begin with a relaxed version of the game: the hints we give to other players are complete information on one unique card. The player get to know everything on that card without any possible ambiguity. However we lose the possibility of giving hints on several cards at the same time.

5.2 Representation of the problem

Because of the complexity and diversity of our states, we chose to explicitly describe the states instead of using a PDDL-like representation. In every state, we need to represent the hands and knowledge of the different players, along with the cards that are either on the table, in the graveyard (discarded cards) or still in the deck. We also need to keep track of the blue and red tokens. The transitions between two states will be represented as actions. Our goal is that our agents come up with sequences of actions leading to a state with a maximal utility (see subsection 5.4).

The tree in figure 5.2 gives an example of a first round of play. This is the tree the first agent generates when it wants to create its initial plan. In other words, the agent simulates the behavior of the other agents, in particular their responses to any action it might take.

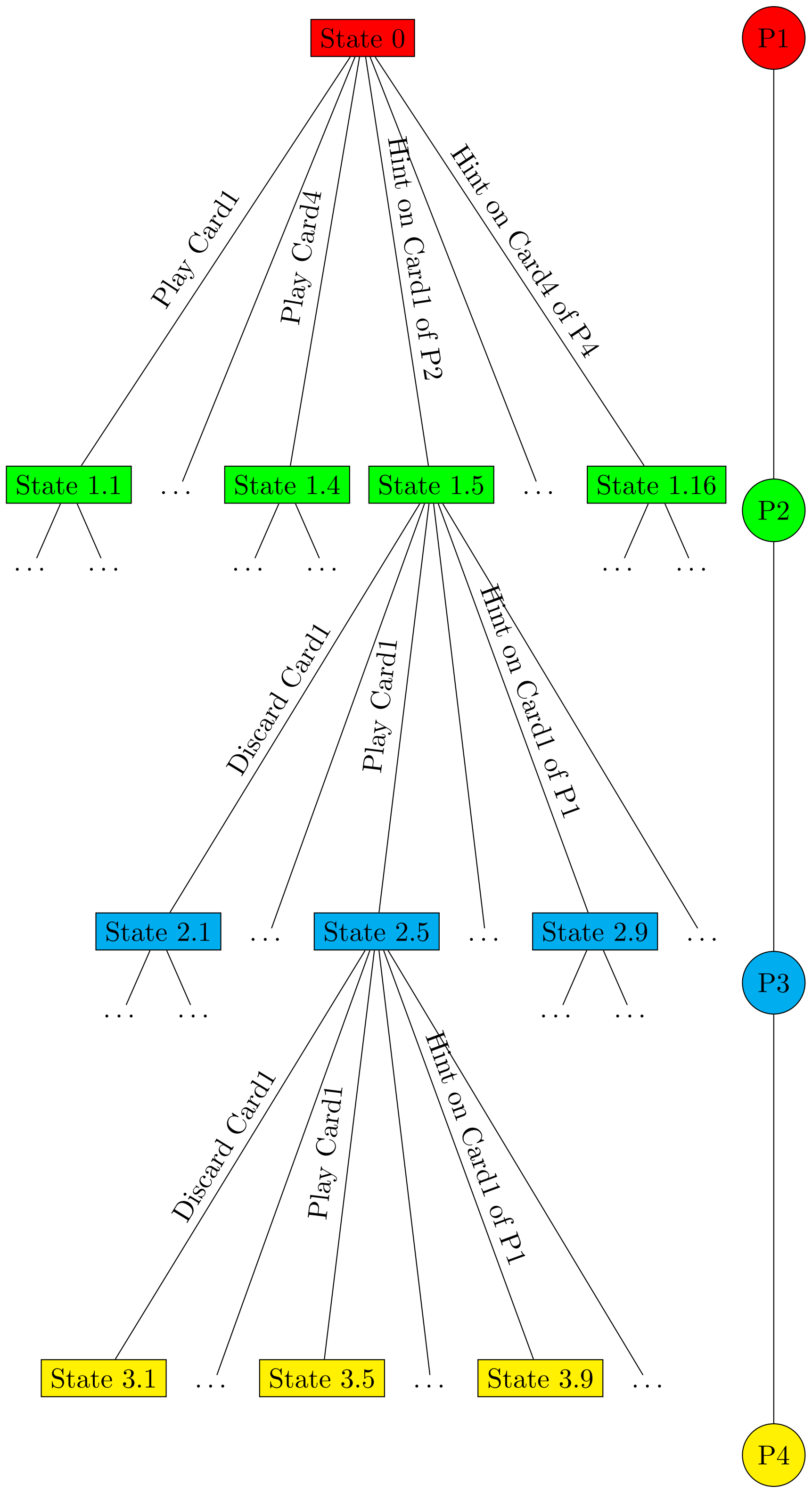

Figure 6: Game tree: players play in turns, different actions lead to different states

5.3 Description of the state

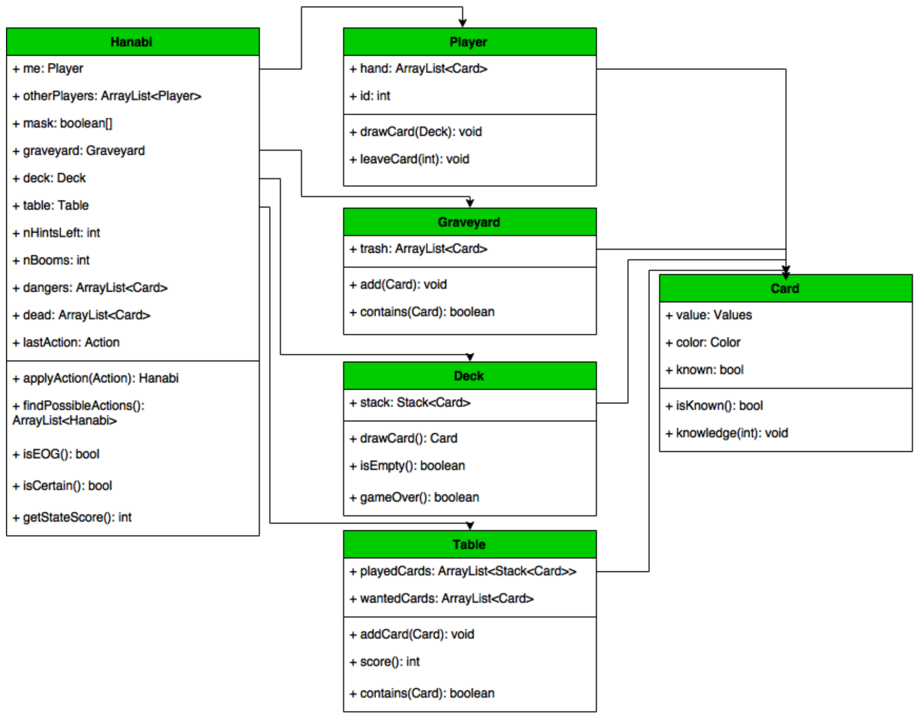

Figure 7: (Partial) class diagram

5.4 Different heuristics

We implemented different heuristics as evaluation functions. The main idea is to compute a score based on the different elements of the game state : number of cards played, number of remaining hints, number of wrongly played/discarded cards The goal of such an heuristics is to give the agent a *knowledge* of how good a state is. We implemented this with an integer score to which several maluses or bonuses could be applied depending on the state of the game:

- A bonus for each card correctly played on the table
- A bonus for each remaining hints
- A malus for each wrongly played cards (cards played on the table but which could not be played, in this case the card is discarded and the players lose one of their three lives)
- A malus for each *dangerous* cards, i.e. each combination color+value with only one card instance not yet discarded.
- A malus for each combination color+value of which corresponding cards have all been discarded.

The evaluation function in the terminal states takes extreme values as high incentives for the agents to end the game with the highest score possible. Indeed:

- The evaluation function returns (Integer.MIN_VALUE + the number of well played cards) when the game ends because the players lost all their lives. This case should be avoided by our agents, and if they cannot they should try to play as many cards as possible before it.

- The evaluation function returns INTEGER.MAX_VALUE if the game ends because the players played all their cards. This is the ultimate goal and the maximum possible score in the game has then been achieved.
- The evaluation function returns the number of well played cards (that is to say the actual score in the game) if the state is an end of game because there is no more cards in the deck.

After observing the behavior of our agents on several states, we also added some other score modificators:

- A malus when the last action has been to give a hint on a card which was already known by the player holding it (to avoid turns where several players were giving the same hint to the same player).
- A malus for each remaining hints at the end of the game as an incentive for our agents not to discard cards when close to the end of the game if there are enough hints.

We then tested our heuristic with many different values (taking the average of results on several tests with different randomly generated decks) for the different maluses and bonuses to finally come up with a final heuristic. Here is how we compute the value of a given game state:

- +100/played cards
- +10/remaining hints
- -100/wrongly played cards
- -20/dangerous cards
- -150/combination color/value whose cards have all been discarded
- No malus when the previous action was to give an hint on a card that was already known by the player. We can explain this result by the fact that the irrelevance of such an action already appears in the other score modificators (giving an already given hint waste a hint (10 points) without leading to an improvement in the score.

5.5 Analysis

With the relaxed game, our players manage to get a score between 16 and 20 points, which is considered “excellent, crowd pleasing”. It runs in 0.4s, which is faster than real players. With the relaxed game rules, our agents perform with a score of 18.8 on average over 25. This is slightly less than the scores good human players can get. We can explain it by the fact that by experience, humans develop more subtle techniques and make a bigger use of logic deduction and context. In particular, human players assume that every clue is optimally intelligent, or at least that every clue is given for a reason. In some cases, a human player C can give a clue to a player B on a not-yet-playable card to make another player A infer something about their own cards (for instance that they have the card that would allow player B to play their card, most probably the most recent card in their hands). By doing that, player C makes two other players play cards by using just one hint. Our current agents do not have that deepness in their reasoning. We then implemented realistic hints (show all cards with a given color or value in one player's hand). To do that, we replaced the action Hint with two new actions : HintColor and HintAction. Card objects have now an attribute possibleCards, a matrix of integers that is updated when looking at other in-game cards and when receiving a hint. Hint is a class in which we store the given color, the given value (one of them is set to Unknown) and a boolean “anti” set to false if the hint means “this card is

hint", and true if it means "this card is not **hint**". The given hints on a card are stored in an ArrayList. With realistic hints, we get scores between 11 and 15, which is considered *honorable* (up to 19 with an optimally shuffled deck), and it runs in 7s. Finally we implemented real players' behavior (Given a hint on several cards, I assume that only my most recently shown card matter, as someone would have tried to tell me if I had a playable card for a long time. Hence I play it, even if I don't have a full knowledge of it. I still keep the given information on my other card, it might be useful later). With some hard-coded behaviors (when given a hint, the player assume that the most recent given card is playable, unless not any playable card has the given attribute, or it is not consistent with what the player already knows), we get scores between 6 and 21, and it runs in 7s. This high range is due to the possibility of giving unwanted hints to other players.

6 Summary

6.1 Conclusions

By carefully choosing a good formalism and representation together with a forward search algorithm with a properly fitted evaluation function, we managed to implement agents performing as well as trained humans. Through our study, we learned the different approaches that one can adopt to tackle the implementation of intelligent agents in a cooperative game with incomplete information.

6.2 Looking for improvements

By carefully choosing a good formalism and representation together with a forward search algorithm with a properly fitted evaluation function, we managed to implement agents performing as well as trained humans. Through our study, we learned the different approaches that one can adopt to tackle the implementation of intelligent agents in a cooperative game with incomplete information.

References

[1] M. de Weerdt, "Introduction to planning in multiagent systems,"

[2] S. M. LaValle, *Planning Algorithms, Chapter 11 (especially 11.1 and 11.7) of.*

[3] H. Osawa, *Solving Hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information.* University of Tsukuba, Japan.