A knowledge-based system for heuristic search in a competitive multiagent game

M. Maliković¹, M. Ćubrilo²

Abstract – In this paper we shall consider possibilities of the heuristic development of a tree of actions and states in a competitive multiagent system. These heuristics are developed on the basis of the agents’ knowledge about the current state of the system and their knowledge concerning the knowledge of other competitive agents. Furthermore, they are based upon deductive conclusions which can be derived by using the available information about the system. The model is based on a simple card game with incomplete information, which means that the developed heuristics actually represent possible strategies used by the player in order to achieve the highest possible win probability. Along with theoretical assumptions, the paper describes a concrete system built in the deductive system Coq for two purposes: the verification of the possibility of implementing the theoretical idea, as well as the statistical measurement of the results achieved by different strategies at different levels of reasoning. For the purpose of reasoning about knowledge, special knowledge matrices have been introduced, which can be used on different levels of nested knowledge. They can also be upgraded on the basis of the available information about the state of the system. They could be used by the players as a source of information for using the chosen strategies.

Keywords: Knowledge-based system, Competitive multiagent game, Heuristic search, Card game, Coq

I. Introduction

Pig is a very simple and enjoyable game for a large number of players. Up to 13 players can take part using a standard 52 card pack, and if two or more packs are used then there can be more players. The players sit in a circle, and are each dealt four cards. The objective is to collect four cards of the same rank, by passing cards one at a time to the neighbour on your left while receiving cards from the player on your right. There are two different ways of playing: with or without a stock pile.

The version of the game without a stock pile requires four cards of one rank for each person playing. For example, with seven players you can use all the aces, kings, queens, jacks, tens, nines and eights from a standard 28 card pack. In this paper, we consider a version of the game with nine cards of three different ranks and with three players. The cards are shuffled and dealt out to the players so that everyone has three cards. All players simultaneously place one unwanted card face down to their left, and then pick up the card that the player to their right has placed. Then they repeat the procedure, and continue to do so until someone collects a three of a kind. There are no turns - the passing happens as fast as the players wish, but during play the cards must be passed one at a time, and you must never have more than three cards in your hand at once. This means that you are not allowed to pick up a new card from your right-hand neighbour until you have reduced your cards by discarding one to your left. During the game, the players can see the order of the opponents’ cards.

Pig, like most games, can be viewed as a competitive multiagent system in which any given agent (player) needs to consider the actions of other agents and the state of the whole system [1]-[4]. In addition, Pig is a game with imperfect information, so the players have to learn some facts during the game and deduce some conclusions from these facts. Also, the players can learn some facts about the opponents’ knowledge and how to use them in making decisions about their own actions.

Similar as in [5], we define the game of Pig as a kind of a search problem with the following components:

- The initial state - distribution of cards but without identifying which player moves first;
- The successor function - returns the new states arising from the previous state;
- The terminal test - determines when the game is over (terminal states);
- The utility function - gives a value for terminal states (in the game of Pig these values can be: one of three players wins or all three players win).
Theoretically, in the game of Pig we can have a search graph instead of a search tree due to the following: the same state can be reached by multiple paths and repeated states are possible. However, we shall see that this will not turn out to be a desirable outcome. A state in the play does not only refer to the distribution of cards among players, but also to the knowledge acquired by players during the play. Due to the players’ reasoning about the knowledge of other players as well as the possibility of upgrading this knowledge, the repeated states are very rare.

Pig differs from “common” competitive games in that the moves of the players do not alternate but occur simultaneously. In the games in which the moves of the players alternate, the move of a player is determined by the move of the player moving in the game before him.

In the case of the game of Pig, it is necessary for players to simultaneously seek the optimum strategy for the game with their own moves and those of their opponents. It can be said that at the moment of making a move, the state of the environment is changing for the player, not only due to their move, but also because of the moves of all the other players.

The Coq system is a computer tool for verifying theorem proofs in higher-order logic. These theorems may concern usual mathematics, proof theory, or program verification. The underlying theory of the Coq system is the Calculus of Inductive Constructions [6]-[7] a formalism that combines logic from the point of view of λ-calculus and typing. Objective Caml is the implementation language for Coq. Depending on the proposition that one wants to prove, the Coq system proposes tools, called tactics [8], to construct a proof, using elements taken from the context, namely, declarations, definitions, axioms, hypotheses, lemmas, and already proven theorems. In addition, the Coq system provides operators, called tacticals [9]-[10] that make it possible to combine tactics and thus build more complex tactics. This paper presents a formal system for heuristic search based on knowledge and different levels of nested knowledge in the game of Pig using Coq.

II. Foundations

Before the basic definitions are introduced, it is necessary to open a new section and load the List module [8], since our model is going to be based on lists:

Section Pig.

Require Import List.

The labels of the players are introduced as p1, p2 and p3, while the colours of the cards are marked as follows:

- H - Heart, C - Club and D - Diamond. The unknown cards, with respect to their initial position, will be marked Xij for i, j ∈ {1, 2, 3}. The colours of the cards are introduced as enumerated inductive type [6].

Inductive card_color : Set :=
H | C | D | X11 | X12 | X13 | X21 | X22 | X23 | X31 | X32 | X33.

In order to be able to compare the values of the cards it is necessary to prove the lemma eq_card_color, which states that two elements of the card_color type can only be equal or different:

Lemma eq_card_color :
forall x y : card_color, (x=y) + (¬x=y).
Defined.

We declare the card distribution cd among the players at any given moment as the following type:

Parameter cd : nat -> list (list card_color).

The nat type, which appears in the declaration above, is in fact the temporal parameter signifying the index number of the round played. On the other hand, the list card_color type actually presents the list of cards in possession of one of the players, while list (list card_color) represents a list of such lists. Let us now state the hypothesis on the initial distribution of cards in moment 0, that is, before the first move is played, and let us state it in the most general fashion, that is, with the assumption that each of the players holds three different cards:

Hypothesis Hcd :
| cd 0 =
| (H :: C :: D :: nil) ::
| (H :: C :: D :: nil) ::
| (H :: C :: D :: nil) :: nil.

The first row in the Hcd hypothesis refers to player p1, the second and the third to players p2 and p3, respectively.

III. Moves and changes in state

Each of the players have their own cards in a given order, which can be seen by the other players. For example, in accordance with the Hcd hypothesis introduced above, it follows that each of the players holds

3 Although it seems unnatural at first sight, it is still necessary to introduce the Xij unknowns as constructors of the card_color type so as to enable the comparison of colours H, C and D with those variables, which we will have to use later.

4 In the Coq system, the first element of a list is indexed, that is, its value is retrieved using the index number 0, not 1, which should be kept in mind when reading the code presented in this paper.

1 http://coq.inria.fr
2 http://caml.inria.fr
cards distributed in the $H, C, D$ sequence. This sequence will be changed during the game. Variables $m1, m2, m3$ will be used to index the numbers of cards which are released by players $p1, p2$ and $p3$ in each round. The ordered triplets $(m1, m2, m3)$ will be referred to as moves.

Before we define the function which calculates the new distribution of cards after each played move we need an auxiliary recursive function [11] which, on the basis of a given linear list of cards, returns a list of the same kind but with the element on the $nth$ position replaced by a new card. That is, the function arguments are the list of cards $l$, the new card $b$ and the index number $n$ of the element replaced by card $b$:

```
Fixpoint card_change (n : nat) (b : card_color) (l : list card_color) :=
match l with
| [] => []
| n :: l' => n :: (card_change (n + 1) b l')
end.
```

The function returning the new distribution of cards after the move $(m1, m2, m3)$ is defined as follows (it is necessary to declare the required parameters before the function itself):

```
Parameter CX : card_color.

Definition cd_new m1 m2 m3 (l : list (list card_color)) :=
  card_change m1 (nth m3 (nth 1 l nil) CX) (nth 0 l nil) ::
  card_change m2 (nth m1 (nth 0 l nil) CX) (nth 1 l nil) ::
  card_change m3 (nth m2 (nth 1 l nil) CX) (nth 2 l nil) :: nil.
```

The axiom which will enable us to perform a multiple iterative sequence of card distribution while increasing the temporal parameter by 1 during each iterative step is introduced as:

```
Axiom AIteration :
forall on : nat,forall C11 C12 C13 C21 C22 C23 C31 C32 C33 : card_color, cd on = (C11 :: C12 :: C13 :: nil) :: (C21 :: C22 :: C23 :: nil) :: (C31 :: C32 :: C33 :: nil) => cd (S on) = cd_new m1 m2 m3 ((C11 :: C12 :: C13 :: nil) :: (C21 :: C22 :: C23 :: nil) :: (C31 :: C32 :: C33 :: nil) :: nil).
```

IV. The end of the game

The game is over when at least one of the players holds three cards of the same colour. The possible endings are the following:

1. Everybody wins
2. Only player $p1$ wins
3. Only player $p2$ wins
4. Only player $p3$ wins

In accordance with this we shall set a goal as follows:

```
Goal exists on : nat,

  nth 0 (nth 0 (cd on) nil) CX = nth 1 (nth 0 (cd on) nil) CX ∧ nth 0 (nth 0 (cd on) nil) CX ∧ nth 2 (nth 0 (cd on) nil) CX ∧
  nth 0 (nth 1 (cd on) nil) CX = nth 1 (nth 1 (cd on) nil) CX ∧ nth 0 (nth 1 (cd on) nil) CX ∧ nth 2 (nth 1 (cd on) nil) CX ∧
  nth 0 (nth 2 (cd on) nil) CX = nth 1 (nth 2 (cd on) nil) CX ∧ nth 0 (nth 2 (cd on) nil) CX ∧ nth 0 (nth 2 (cd on) nil) CX ∧
  nth 0 (nth 0 (cd on) nil) CX = nth 1 (nth 0 (cd on) nil) CX ∧ nth 0 (nth 0 (cd on) nil) CX = nth 2 (nth 0 (cd on) nil) CX ∧
  nth 0 (nth 1 (cd on) nil) CX = nth 1 (nth 1 (cd on) nil) CX ∧ nth 0 (nth 1 (cd on) nil) CX = nth 2 (nth 1 (cd on) nil) CX ∧
  nth 0 (nth 2 (cd on) nil) CX = nth 1 (nth 2 (cd on) nil) CX ∧ nth 0 (nth 2 (cd on) nil) CX = nth 2 (nth 2 (cd on) nil) CX.
```

A goal set in such a way is actually a disjunction of four statements in the same sequence which was used above to give the possible game endings. In each of the four given cases, the game has to be stopped. The way to do this for each of the cases will be described in Section VI.

V. Players’ knowledge matrices

In this section we shall define the matrices of the players’ knowledge about the distribution of cards in a given moment. In general, these matrices are equivalent to the matrix of card distribution in the $Hcd$ hypothesis, which means that their values correspond to those given in the respective fields of the $Hcd$ matrix. The difference is that the players are acquainted with some values of the matrix while unfamiliar with other values, and their knowledge varies during the game. It is clear that a different knowledge matrix is attributed to each player. The matrix elements known by the players are displayed using their concrete values ($H, D$ or $C$), while the elements unknown to the players are replaced by variables $Xi j$ for $i, j \in \{1, 2, 3\}$, depending on the initial position of a card.

Before we define the described knowledge matrix, we shall introduce the concept of knowledge as the following inductive type [12]:

```
Inductive K (i : nat) (on : nat) (f : Prop) :=
  know : f -> K i on f -> K i on f.
```

In the definition above, parameter $i$ of the $nat$ type denotes the index number of a player, parameter $on$ of the $nat$ type is a temporal parameter, while parameter $f$ of the $Prop$ type denotes the proposition known by player $i$ in moment $on$.

In moment 0, that is, before any action has been performed by the players, the players are acquainted only with the cards in their hands, so this hypothesis will be introduced as follows:
Parameter cdk : list (list card_color).

Hypothesis HK1 : K 1 0 (cdk = ((H :: C :: D :: nil) :: (X21 :: X22 :: X23 :: nil) :: (X31 :: X32 :: X33 :: nil) :: nil)).

Hypothesis HK2 : K 2 0 (cdk = ((X11 :: X12 :: X13 :: nil) :: (H :: C :: D :: nil) :: (X31 :: X32 :: X33 :: nil) :: nil)).

Hypothesis HK3 : K 3 0 (cdk = ((X11 :: X12 :: X13 :: nil) :: (X21 :: X22 :: X23 :: nil) :: (H :: C :: D :: nil) :: nil)).

After each move, it is necessary to apply an axiom similar to the previously introduced $A_{\text{iteration}}$ axiom to hypotheses HK1, HK2 and HK3, which is defined as follows:

Axiom $A_{\text{iteration}}_K$ :
forall p on : nat, forall C11 C12 C13 C21 C22 C23 C31 C32 C33 : card_color, K p on (cdk = (C11 :: C12 :: C13 :: nil) :: (C21 :: C22 :: C23 :: nil) :: (C31 :: C32 :: C33 :: nil) :: nil) -> K p (S on) (cdk = cd_new m1 m2 m3 ((C11 :: C12 :: C13 :: nil) :: (C21 :: C22 :: C23 :: nil) :: (C31 :: C32 :: C33 :: nil) :: nil)).

After the exchange of cards, the application of the $A_{\text{iteration}}_K$ axiom for each player in the knowledge matrix preserves the information on the whereabouts of their card, since they are known to them after the exchange. However, each player also knows which card they have received, that is, they know its concrete value. This means that after the exchange it is necessary to replace eventual unknowns $X_{ij}$ with concrete values in the row of each knowledge matrix referring to those cards held by a respective player. A simple way to do this is to replace the whole row with its corresponding row from the $H_{cd}$ hypothesis, which can be achieved by applying the following axiom (only the axiom for player $p1$ is given, the other ones are analogous):

Axiom $A_{\text{get card}}_1$ :
forall on : nat, forall C11 C12 C13 C21 C22 C23 C31 C32 C33 : card_color, K 1 on (cdk = (C11 :: C12 :: C13 :: nil) :: (C21 :: C22 :: C23 :: nil) :: (C31 :: C32 :: C33 :: nil) :: nil) -> K 1 on (cdk = nth 0 (cd on) nil) :: (C21 :: C22 :: C23 :: nil) :: (C31 :: C32 :: C33 :: nil) :: nil).

Apart from what has been already mentioned, each player also knows the whereabouts of a card exchanged by other players if the card itself is known to them. It is easily understandable that knowledge matrix transformations contain this data as well.\footnote{The knowledge matrices also contain data about the movement of cards, even in the case of a concrete value of a card being unknown to a player. For example, although the card hiding behind the $X_{12}$ variable is not known to player $p1$, they know that the card was previously held by player $p2$, and is currently held by player $p3$. Although this sort of complex reasoning is not the subject of this paper, its capacity to provide the players with valuable insights can be proven.}

VI. Development of moves and game states

Before the first move none of the players know anything, which implies that no strategy is possible and that the players play by chance. Therefore we can assume that, for example, the move (0, 1, 2) has been played. We shall introduce this case in the form of a hypothesis which will be deleted from the context after the first move has been played:

Hypothesis Hm1 : m1=0.
Hypothesis Hm2 : m2=1.
Hypothesis Hm3 : m3=2.

Our requirements for the description of further possible moves are satisfied by a small subset of natural numbers, in this case \{0, 1, 2\}. Therefore we shall use the annotated inductive type for this purpose \[9\]. That is, we are defining the move type as follows:

Inductive move : nat -> Prop :=
  | move0 : move 0
  | move1 : move 1
  | move2 : move 2.

In order to execute inductive development of possible moves, that is, ordered triplets (0, 0, 0), (0, 0, 1), (0, 0, 2), (0, 1, 0), ..., (2, 2, 2), it is necessary to introduce the following axioms:

Axiom Am1 : move m1.
Axiom Am2 : move m2.
Axiom Am3 : move m3.

The initial tactics that are applied will differ slightly from the tactics applied afterwards, due to the stated hypotheses $H_{m1}$, $H_{m2}$ and $H_{m3}$. In any case, the $Ltac$ function \[10\], which will be employed only for the first iteration, is defined as follows:\footnote{Initial $Ltac$ functions introduced in this section will be elaborated in further sections.}

Ltac Play0 :=
  apply $A_{\text{iteration}}$ in $H_{cd}$;
  apply $A_{\text{iteration}}_K$ in HK1;
  apply $A_{\text{iteration}}_K$ in HK2;
  apply $A_{\text{iteration}}_K$ in HK3;
  rewrite Hm1 in * ;;
  rewrite Hm2 in * ;;
  rewrite Hm3 in * ;;
  clear Hm1 Hm2 Hm3;
  compute in * ;;
  apply $A_{\text{get card}}_1$ in HK1;

...
The *Ltac* function *Play*, which will be applied after the initial application of function *Play0* differs from this function in that it develops moves, that is, numbers *m1*, *m2* and *m3* inductively. This means that function *Play* will use the *induction Am1* tactics (and the analogous ones for numbers *m2* and *m3*), instead of the *rewrite Hm1 in * |- tactics. Furthermore, it means that the *clear Hm1 Hm2 Hm3* tactic will not be used, since these hypotheses do not exist in this context after the *Play0* function has been applied.

Let us now explain the order of the application of the introduced *Ltac* functions and the required tactics. As already mentioned, the first function to be applied after the goal has been set is the *Play0* function:

Play0;

At this moment it is not necessary to try to prove the goal, as we know that none of the players could have possibly won as yet. Only after the first application of the *Play* function would it have been possible for a player to win, so we can try to prove the subgoals. In any case, after:

Play;

the goal set develops into 3^3=27 subgoals. As already mentioned, the conclusion of the goal set and now appearing equally in all the derived subgoals is a disjunction of four statements, the first of them stating that all three players have won, the other three stating that either player *p1*, player *p2* or player *p3* has won. In cases in which some of these statements have been proven, the proof of the subgoal belonging to it will be saved as a separate lemma whose prefix will be *all_win, p1_win, p2_win* or *p3_win*, depending on the winner. This means that in case of player *p1* winning *n* subgoals we shall have *n* lemmas titled *p1_win, p1_win0, p1_win1, ..., p1_win(n-2)*. All of this can be achieved using the following tactics:

try (exists 3;rewrite Hcd;compute;(left;abstract tauto using all_win) || (right;left;abstract tauto using p1_win) || (right;right;left;abstract tauto using p2_win)) || (right;right;right;abstract tauto using p3_win));

Further development of the movement tree could proceed in an analogous fashion:

Play;

However, after the fourth move is played, we would already have too many subgoals and in such cases the Coq system would already show computational problems.  

VII. Reducing the search space

In a lot of games we can reduce the search space by symmetry reduction. In [14] we can see how it can be done in the game of tic-tac-toe. In our case it can be said that if one player holds two equal cards, the situation is analogous and the choice of the card made by the player is irrelevant. As we shall see later, this is not actually true, since such solutions are heuristic solutions. As we know, heuristics can lead a search algorithm to a suboptimal solution or fail to find any solution at all. Nevertheless, for now we shall satisfy ourselves with this line of reasoning.

As the basis for the reduction, we are going to upgrade the system with inductive definitions and axioms of a form:

Inductive moveij : nat -> Prop := | moveij_1 : moveij i | moveij_2 : moveij j.

Axiom Amk_ij : moveij mk.

for *i, j* = 0, 1, 2, *i ≠ j*, *k* = 1, 2, 3.

In order to reduce the search space using the given definitions and axioms, and to do it so that in the case of a player holding two equal cards with the lower index number is chosen, the following tactics can be used in the *Play* function instead of tactics *induction Am1; induction Am2; induction Am3* (we shall only give the case for player *p1*, the tactics for players *p2* and *p3* being analogous):

try (exists 3;rewrite Hcd;compute;(left;abstract tauto using all_win) || (right;left;abstract tauto using p1_win) || (right;right;left;abstract tauto using p2_win)) || (right;right;right;abstract tauto using p3_win));
We can consider this way of playing the game to be a game without strategy and denote it WS. It is clear that none of the players will have a greater win probability, that is, a greater number of branches in the search tree finishing with their win if all the players play the game in this way.\(^{10}\)

**VIII. Upgrading knowledge matrices**

At the beginning of this section it is necessary to introduce the `count_cards_row` function, which will give us the number of cards of a specific colour in a particular row of the list of card lists, and the `count_cards` function, which will give us the same number, but for the whole list of card lists:

```
Fixpoint count_cards_row (l : list card_color) (x : card_color) {struct l} : nat :=
| match l with
|   | nil => 0
|   | y :: rest =>
|   | let n := count_cards_row rest x in
|   | if eq_card_color y x then S n else n
| end.
```

Definition count_cards (l : list (list card_color)) (x : card_color) :=
```
count_cards_row (nth l 0 nil) x + count_cards_row (nth l 1 nil) x + count_cards_row (nth l 2 nil) x.
```

It can be said that the knowledge possessed by the players before the game has started as well as during its course is:

1. There are exactly three cards of the Heart, Diamond and Club colour
2. None of the players holds three equal cards\(^{11}\)

From this knowledge the players can, if they possess some additional information, come to various conclusions using deductive reasoning. It is clear that some of these conclusions are already made possible by the knowledge of the cards held by the player themselves. One such conclusion can be: “If I hold two Hearts and one Diamond, it follows that only one of the opponents holds a Heart, and only one at that.” However, conclusions like this are too general and it is difficult to implement them into the system itself. Therefore, we shall concentrate on simpler conclusions leading directly to new concrete knowledge about the distribution of cards and introduced formally using the transformation of the individual players’ knowledge matrix.

Let us consider the possibility of upgrading the knowledge matrix in the example of player `p1`. Fig. 1 shows the general development of possible knowledge matrices independently of the moves played. “O” denotes the position of the cards known to player `p1` and “X” those of the cards that are unknown to them. The possibilities in Figure 1 are shown in a reduced form, without taking into consideration the position of the known cards. What is important is the number of the known cards, and the conclusion will be deduced in a general manner in the code itself.

After the first move, each player knows the whereabouts of the card they have passed to the next player. However, this knowledge does not make it possible to upgrade the knowledge matrix in any way. It only enables the players to come to various deductive conclusions consisting of disjunctions, conjunctions and negations. As already mentioned, the introduction of such statements in a system would significantly slow down the computation, and it would be difficult to operate on them due to the great number of different possibilities.

After the second move, two scenarios are possible. In the first case, the card is simply passed on to player `p2` by player `p1` is passed by player `p2` to player `p3`, which means that player `p1` now knows that this card is held by player `p3`. In the second case, a card unknown to player `p1` is passed on to player `p3` by player `p2`, meaning that player `p1` now knows two cards held by player `p2`. Considering all the possible variants of cards denoted by “O” after the second move, the following rule can be stated:\(^{12}\)

P1. If matrix `2b` contains one triplet and one pair of cards of the same colour denoted by “O”, this means that the second row of the third column contains a card of a third colour. The same rule can be applied to matrix `3a`, with the provisions above now referring to the second instead of the third row and the additional knowledge of a card in the second row being of no importance. It can also be applied to matrix `3b`, independently of an additional known card in the third row. And finally, the rule also applies to matrix `4b`, for the second and the third rows, respectively.

We shall implement rule P1 in the form of an axiom.

---

\(^{10}\) It is important to note that the reduction used in this section does not lead to the loss of the most efficient simple strategies, that is, those strategies in which players holding two equal cards choose the third card.

\(^{11}\) Since this game is actually a discrete sequence of simultaneous actions by all players, we assume that the player who wins is going to announce their victory immediately. Further reasoning by other players is not necessary, so the situation in which all three cards are held by one player is considered to be impossible.

\(^{12}\) Somebody could ask why we did not simply assume a finite set of possible combinations of known and unknown cards and deduce the rules from this, instead of showing which situations can occur in specific moves. The answer is that in this way certain rules will be applied only after the moves which can lead to a certain situation. This will save us computing resources.

\(^{13}\) For example, cards H, H, H, C, C.
Before this is done, it is necessary to import the knowledge axiom, that is, the so-called T-axiom from the multimodal epistemic logic $S5_m$, which is generally defined as $K_i \phi \rightarrow \phi$ (if proposition $\phi$ is known to agent $i$, then the proposition is valid, in other words, it is impossible for an agent to know a false proposition) [15]-[18]. This axiom will enable us to “pull out” a proposition known by a player as an independent proposition from the knowledge operator. The knowledge axiom is defined as (in the context of our paper):

\[
\text{Axiom Knowledge : } \forall i : \mathbb{N}, \forall \phi : \text{Prop}, K_i \phi \rightarrow \phi.
\]

An example of an axiom implemented by rule $P1$, which can be applied to matrices in conjunction with the second row, that is, to the cards of player $p2$, can be defined as follows:

\[
\text{Axiom TMP1p1_2 : } \forall i : \mathbb{N}, \forall \phi : \text{Prop}, K_1 \phi \rightarrow \text{count\_cards\_row}(\text{nth} 0 \text{ cdk nil}) C + \text{count\_cards\_row}(\text{nth} 1 \text{ cdk nil}) D + \text{count\_cards\_row}(\text{nth} 1 \text{ cdk nil}) D = 5.
\]
The first on of ending is clear that the players do not benefit players, adding cards of the same colour movement of the cards of the same colour among the willingness) of the player to memorize cards, follow the strategy but rather a reflection of the ability (and application of knowledge matrices with 6 and more known cards can appear in the context only after the third move, making upgrade a knowledge matrix sufficiently for the been made, because a successful application of UKM.

However, it is clear that the players do not benefit from the matrix upgrade if the knowledge acquired from this reasoning is not used within a specific game strategy.

 IX. The game involving knowledge of other players’ cards

There are different ways to play the game depending on the players’ knowledge about the other players’ cards. If we focus on player p1, then one of the ways of playing the game is based on player p2 being passed a card that is the least convenient for them. Let us notice that such a game strategy makes sense for player p1 only in the case when all the three cards they have are different. In case of p1 holding two equal cards X and one card Y, avoiding playing card Y would necessarily lead to a greater probability of losing the game, that is, there is a lesser probability of winning it. In such a case, the best score is still yielded by the WS game. However, in case player p1 has three different cards and p1 knows that p2 has two equal cards, they will not give them the third card of the same kind. If the player cannot come to this conclusion, it means they are playing without a strategy.

The tactics for choosing the cards in this way, referring to player p1, can be created as follows:

try (apply TMP1\_p1\_2 in HK1;[rewrite Hcd in HK1;compute in HK1 | apply Knowledge in HK1;rewrite HK1;compute;tauto]);

As already stated, the application of rule P1 can succeed only after the second move, that is, after the first application of the Play function. Further, it is better to try to apply this rule after it has already been tried so as to prove the subgoals leading to the win of a player, as in this case there are fewer subgoals left.

By careful observation it can be concluded that there is only one rule which can be applied to matrices containing 6 or more known cards:

P2. If a knowledge matrix contains 6 or more known cards and if two full triplets of some of the card colours are to be found among them, then there is a third colour card behind each unknown card.

Rule P2 can be implemented using one axiom only:

Axiom TMP2 : forall i on : nat, forall m : list (list card_color), K i on (cdk = m) ->

count\_cards m H + count\_cards m C = 6 V

count\_cards m H + count\_cards m D = 6 V

count\_cards m C + count\_cards m D = 6 -> K i on (cdk = cd on).

The following tactics can be employed to try to apply this axiom on knowledge matrices (for example, the knowledge matrix of player p1):

try (apply TMP2 in HK1;[rewrite Hcd in HK1 | apply Knowledge in HK1;rewrite HK1;compute;tauto]);

The application of rule P2 to the knowledge matrices of players p2 and p3 is analogous. An attempt to apply rule P2 should be made after an attempt to apply P1 has been made, because a successful application of P1 can upgrade a knowledge matrix sufficiently for the application of P2 to be also successful. Further, knowledge matrices with 6 and more known cards can appear in the context only after the third move, making the application of P2 before this meaningless.

Upgrading knowledge matrices is not a playing strategy but rather a reflection of the ability (and willingness) of the player to memorize cards, follow the movement of the cards of the same colour among the players, adding cards of the same colour and using this combined knowledge to deduce conclusions. Admittedly, these are (at least for humans) quite complex operations. Therefore, in this paper we shall tentatively introduce upgrading knowledge matrices as a way of playing denoted UKM.

match goal with

\[
\begin{align*}
[Hcd : cd ?on = cd\_new ?m1 ?m2 ?m3 ((?X :: ?Y :: ?X :: nil) :: ?L2 :: ?L3 :: nil)] & \Rightarrow \text{ induction } \text{Am1}_1 \\
[Hcd : cd ?on = cd\_new ?m1 ?m2 ?m3 ((?X :: ?Y :: ?X :: nil) :: ?L2 :: ?L3 :: nil)] & \Rightarrow \text{ induction } \text{Am1}_1 \\
[Hcd : cd ?on = cd\_new ?m1 ?m2 ?m3 ((?Y :: ?X :: ?X :: nil) :: ?L2 :: ?L3 :: nil)] & \Rightarrow \text{ induction } \text{Am1}_0 \\
1 & \Rightarrow \text{ match goal with

\begin{align*}
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_1 \\
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_1 \\
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_0 \\
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_0 \\
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_0 \\
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_0 \\
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_0 \\
[\text{HK1} : K 1 ?on (cdk = cd\_new ?m1 ?m2 ?m3 ((?A :: ?B :: ?C :: nil)))] & \Rightarrow \text{ induction } \text{Am1}_0 \\
\end{align*}
\]

match goal with
Naturally, tactics referring to this way of playing are analogous for players’ \( p_2 \) and \( p_3 \). This strategy can be called Contra First Opponent, and we will label it CFO.

If compounded by upgrading the \( p_1 \) player’s knowledge matrix, the CFO strategy increases the probability of their win if players \( p_2 \) and \( p_3 \) play without a strategy. If all three players use the CFO strategy without upgrading the knowledge matrix, then all three players have an equal win probability. If player \( p_1 \) is the only one to upgrade the knowledge matrix, then the probability of their win is further increased at the expense of other players, but other players are still able to compensate for this if they upgrade their knowledge matrices. This poses the question of the possibility of the player \( p_1 \) to increase their win probability in such a situation. We shall give a possible answer to this question in Section X.

X. Reasoning on other players’ knowledge

The player who has received a card also knows that other players, who possessed this card at any previous moment, know that this card is currently held by them. The CFO strategy can be improved if player \( p_1 \), in case of having to choose arbitrarily between the cards in their possession, chooses a card known by their opponents to be in a certain position. The basic idea of this strategy is to make it as difficult as possible for the opponents to upgrade their knowledge matrices.

Let us now introduce this higher level of reasoning about knowledge, that is, the knowledge about the other players’ knowledge.

Naturally, before the first move is played, a player does not know anything about the knowledge of other players concerning their cards. This fact can be introduced as a hypothesis for moment 0, which would state the following about the knowledge of player \( p_1 \) about the knowledge of players’ \( p_2 \) and \( p_3 \):

Hypothesis HK1K2 : K 1 0 (K 2 0
\( \text{(cdk = (X1 :: X12 :: X13 :: nil))} \):
\( \text{(H :: C :: D :: nil))} \):
\( \text{(X31 :: X32 :: X33 :: nil :: nil))} \).

Hypothesis HK1K3 : K 1 0 (K 3 0
\( \text{(cdk = (X1 :: X12 :: X13 :: nil))} \):
\( \text{(X21 :: X22 :: X23 :: nil :: nil))} \).

In order to be able to upgrade the matrices from the hypotheses introduced above, on the basis on the moves played, the following rule must be observed: If player \( p_i \) gets a card from player \( p_j \) in moment \( on \), then player \( p_i \) knows that player \( p_j \) knows that this card is held by them as well as the position in which it is held. For the purpose of the application of this rule we need axioms which will be built into the existing system and are defined as follows (only those axioms are given which can be applied to the hypothesis about the knowledge of player \( p_1 \) on the knowledge of player \( p_2 \)):

Axiom AK1K2 :
\[ \forall on : \text{nat}, \forall C11 C12 C13 C21 C22 C23 C31 C32 C33 : \text{card_color}, \]
\[ \text{K 1 on (K 2 on (cdk = (C11 :: C12 :: C13 :: nil)) :: (C21 :: C22 :: C23 :: nil))} \]
\[ \Rightarrow (C31 :: C32 :: C33 :: nil :: nil)) \.

Axiom A_get_card_2_KK :
\[ \forall p : \text{nat}, \forall C11 C12 C13 C21 C22 C23 C31 C32 C33 : \text{card_color}, \]
\[ \text{K p on (K 2 on (cdk = (C11 :: C12 :: C13 :: nil)) :: (C21 :: C22 :: C23 :: nil))} \]
\[ \Rightarrow (C31 :: C32 :: C33 :: nil :: nil)) \.

The new tactics used for the application of these axioms are:

apply AK1K2 in HK1K2;
apply A_get_card_2_KK in HK1K2;

The strategy described at the beginning of this section can be used in case of the CFO strategy giving no results, and in that case, if we observe the tactics for implementing the CFO strategy for player \( p_1 \), we can conclude the following:

1. If player \( p_1 \) has two equal cards, then instead of an arbitrary choice between them a choice should first be made on the basis of the following priorities:
   a. The card known by player \( p_1 \) to be already known by both opponents
   b. The card known by player \( p_1 \) to be known by at least one of the opponents.

2. If player \( p_1 \) has three different cards, and the CFO strategy gives no results, then before making an arbitrary choice, they should try to choose two cards, basing their decision on the following priorities:
   a. Two cards known by player \( p_1 \) to be already known by both opponents
b. Two cards known by player \( p_1 \) to be already known by one opponent, and at least one of them known by the other opponent

c. Two cards known by player \( p_1 \) to be already known by one opponent

d. Two cards, one of them known by player \( p_1 \) to be already known by one opponent, and the other by the other opponent.

By taking into consideration the criteria introduced above, the \( \text{CFO} \) strategy for player \( p_1 \) can be improved by tactics which due to their length cannot be showed in this paper. Such a playing strategy can be called \textit{Contra Upgrading Knowledge Matrices} and can be denoted \( \text{CUKM} \). Player \( p_1 \), when using the \( \text{CUKM} \) strategy combined with \( \text{UKM} \), that is, with the upgrading of their knowledge matrices, and under the assumption of players \( p_2 \) and \( p_3 \) playing without a strategy, further increases their win probability compared to the win probability when playing using the \( \text{CFO} \) strategy. On the other hand, where players’ \( p_2 \) and \( p_3 \) are concerned, using the \( \text{CFO} \) strategy combined with the upgrading of their knowledge matrices will yield similar results as to when the game is played using no strategy at all.

\section{XI. Conclusion}

Much of the research in game theory and its application is related to those situations in which an agent may know something about what the other agents know. Our aim was to propose its practical implementation grounded in theory which will enable players to compute their higher-order knowledge in any state of the game and to use some facts about the opponents' knowledge in making decisions about their own actions. We have shown that Coq - a formal proof management system, its set of tactics for providing proofs as well as operators called tacticals for combining tactics can be used for reasoning about such higher-order agents’ knowledge and for building the state space in complex knowledge-based multiagent systems.

Some interesting questions for further research are: whether this approach can give an answer to the question of what the optimal strategies are for players in knowledge-based card games and whether this approach can be extended to incorporate common and distributed knowledge among the players.

\section{References}


