CHAPTER 17

Games With Incomplete Information

17.1 Introduction

Our input language GDL has been designed with the assumption that after every round the players are informed about each others' moves. The effects of these moves are completely and unambiguously determined by the game rules, so that players can always compute the subsequent game state from the current one. Given that the players have full knowledge of the initial position, they effectively have complete knowledge of the state throughout a game.

This is adequate for classical board games like Chess, Checkers, and Go. But many games of interest use moves with indeterminate effect, like rolling a die. Others are characterized by incomplete and asymmetric information. In the chess variant known as *Kriegspiel*, fro example, the two players only can see the pieces of their own color. Most card games combine randomized moves (shuffling) with information asymmetry (you can't see cards dealt to the other players). The poker variant Texas hold'em is a point in case.

Truly general game playing therefore requires the extended game description language GDL-II, which enables des descriptions of games with nondeterministic moves and where players have incomplete information about the game state.

17.2 GDL-II

GDL-II stands for "GDL for games with incomplete information." By incompleteness we mean that players do not know the full game. Mathematical game theorists draw a finer distinction between what they call imperfect-information games and those of incomplete information. We do not distinguish between the two and just mention that both can be modeled in GDL-II.

The extended game description language uses the three additional keywords described below.

`random`
: means a pre-defined role that moves purely randomly.

`percept(`*r*`,`*p*`)`
: means that *p* is a percept of player *r* in the game.

`sees(`*r*`,`*p*`)`
: means that player *r* perceives *p* in the next state.

By definition, the role of the random player is to always choose with uniform probability among its legal moves. This can be used to model nondeterministic actions as moves whose actual effect depends on the move simultaneously chosen by the game-independent role `random`.

The keyword `percept(r,p)` is used to define all possible percepts for a player in the same way as GDL-keyword `input(r,a)` describes the range of available moves. Percepts can be anything from the move by another role to a specific state feature such as the value of a card dealt face-down to the player. Even communication between players, both private and public, can be described with the help of percepts that are triggered by certain moves.

What a player actually perceives is described by the predicate `sees(r,p)` and generally depends on both the current state and a joint move, much like GDL-keyword `next(f)` does. Percepts are, by definition, private. Hence, they can be used to model games with imperfect and asymmetric information.

Regular GDL games can be simulated in GDL-II by adding the general game rule

```
sees(P,move(Q,M)) :- role(P) & does(Q,M)
```

With this clause, all players are informed about each others' moves in every round. This information suffices to maintain complete knowledge of the position throughout the game, just as in regular GDL.

17.3 Blind Tic-Tac-Toe

As an example of how to describe imperfect-information games in GDL-II, let us look at a variation of Tic-Tac-Toe where, much like in Kriegspiel, the players don't get to see each others' moves. Of course it may then happen that a player intends to mark a cell that's already been occupied. In that case, the move shall have no effect but the player will be informed about it. To make the game fairer, both players mark concurrently. If they happen to choose the same cell at the same time, then the toss of a coin determines who is successful.

To begin with, our new two-person game, which we call Blind Tic-Tac-Toe, features three roles, where the new random player is needed to simulate the coin toss.

```
role(white)
role(black)
role(random)
```

Note that GDL-II requires you to declare `random` as a role whenever you want to use it in a game, even though it is a pre-defined one. This guarantees upward compatibility between GDL and GDL-II.

The initial state is similar to classical Tic-Tac-Toe (see Chapter 2) but without the state feature to indicate whose turn it is, which is no longer needed now that white and black move concurrently. A new feature records every attempt by a player to mark a cell. Players will only be allowed to try each cell once, which will help to ensure that the game terminates.

```
index(1)
index(2)
index(3)

base(cell(M,N,x)) :- index(M) & index(N)
base(cell(M,N,o)) :- index(M) & index(N)
base(cell(M,N,b)) :- index(M) & index(N)

base(tried(white,M,N)) :- index(M) & index(N)
base(tried(black,M,N)) :- index(M) & index(N)

init(cell(1,1,b))
init(cell(1,2,b))
init(cell(1,3,b))
init(cell(2,1,b))
init(cell(2,2,b))
init(cell(2,3,b))
init(cell(3,1,b))
init(cell(3,2,b))
init(cell(3,3,b))
```

Next, we define legality in Blind Tic-Tac-Toe. In each round, both white and black may choose any cell that they have not already tried. At the same time a coin toss is simulated, the result of which is used to break the tie in case both players attempt to mark the same cell. Accordingly, we call this random move a tiebreak.

```
input(white,mark(M,N)) :- index(M) & index (N)
input(black,mark(M,N)) :- index(M) & index (N)

input(random,tiebreak(x))
```

```
input(random,tiebreak(o))

legal(white,mark(X,Y)) :-
  index(X) &
  index(Y) &
  ~true(tried(white,X,Y))

legal(black,mark(X,Y)) :-
  index(X) &
  index(Y) &
  ~true(tried(black,X,Y))

legal(random,tiebreak(x))
legal(random,tiebreak(o))
```

Next, we look at the new update rules for the game. Any try to mark a cell is recorded. A cell is marked with an "x" or an "o" if the corresponding player chooses that cell and if the cell is blank and not simultaneously targeted by the other player. If both players aim at the same cell, then the cell ends up being marked according to the result of the random tie-breaking move. Finally, if a cell contains a mark, then it retains that mark on the subsequent state; and if a cell is blank and neither player attempts to mark it, then it remains blank.

```
; any new attempt to mark a cell is recorded
next(tried(W,M,N)) :-
  does(W,mark(M,N))

; all recorded attempts are remembered
next(tried(W,M,N)) :-
  true(tried(W,M,N))

; white is successful in marking a blank cell
; when black moves in a different column
next(cell(M,N,x)) :-
  does(white,mark(M,N)) &
  true(cell(M,N,b)) &
  does(black,mark(J,K)) &
  distinct(M,J)

; white is successful in marking a blank cell
; when black moves in a different row
next(cell(M,N,x)) :-
  does(white,mark(M,N)) &
  true(cell(M,N,b)) &
  does(black,mark(J,K)) &
  distinct(N,K)

; black is successful in marking a blank cell
; when white moves in a different column
next(cell(M,N,o)) :-
  does(black,mark(M,N)) &
  true(cell(M,N,b)) &
  does(white,mark(J,K)) &
  distinct(M,J)

; black is successful in marking a blank cell
; when white moves in a different row
next(cell(M,N,o)) :-
  does(black,mark(M,N)) &
  true(cell(M,N,b)) &
  does(white,mark(J,K)) &
  distinct(N,K)

; if both players aim at the same cell, then that cell
; gets marked by the result of the random tiebreak move
next(cell(M,N,W)) :-
  true(cell(M,N,b)) &
  does(white,mark(M,N)) &
  does(black,mark(M,N)) &
```

```
  does(random,tiebreak(W))

; markings are forever
next(cell(M,N,x)) :-
  true(cell(M,N,x))

next(cell(M,N,o)) :-
  true(cell(M,N,o))

; a cell remains blank if no player attempts to mark it
next(cell(M,N,b)) :-
  true(cell(M,N,b)) &
  ~marked(M,N)

marked(M,N) :-
  does(W,mark(M,N))
```

GDL-II is based on the assumption that players are no longer automatically informed about any other players' moves. Thus, without additional hints our Blind Tic-Tac-Toe players would be completely oblivious as to whether any of their attempts to mark a cell was successful. According to the following rules, which define the players' percepts, they are provided with exactly that but no more information.

```
percept(white,ok)
percept(black,ok)

; players get ok when they mark a blank cell
; in a different column from where their opponent moves
sees(R,ok) :-
  does(R,mark(M,N)) &
  true(cell(M,N,b)) &
  does(S,mark(J,K)) &
  distinct(M,J)

; players get ok when they mark a blank cell
; in a different row from where their opponent moves
sees(R,ok) :-
  does(R,mark(M,N)) &
  true(cell(M,N,b)) &
  does(S,mark(J,K)) &
  distinct(N,K)

; white gets ok when he marks a blank cell
; and the random tiebreak went to his side
sees(white,ok) :-
  does(white,mark(M,N)) &
  true(cell(M,N,b)) &
  does(random,tiebreak(x))

; black gets ok when he marks a blank cell
; and random tiebreak went to his side
sees(black,ok) :-
  does(black,mark(M,N)) &
  true(cell(M,N,b)) &
  does(random,tiebreak(o))
```

By these rules a player sees "`ok`" after attempting to mark a cell that was indeed blank and that was not simultaneously targeted by the opponent. The very same will be seen by the player in whose favor the tie was broken, provided again that the cell being aimed at was empty. Since the percepts are identical in both cases, players cannot distinguish between them. Hence, they will not know, for example, if their opponent attempted (and failed) to mark the same cell at the same time. Even when both cases apply together, the percept will not change.

Still, the players obtain enough information to infer which of the board's cells carry their own marks. In turn, this allows them to determine exactly their legal moves according to the rules from above. Generally speaking, a good GDL-II game description should always provide players with

sufficient information for them to know their legal moves and also to know whether a game has terminated and what their result is.

In our example game, the absence of the percept "ok" tells players that they were unsuccessful in their attempt to mark a cell. A smart player can even conclude that the cell in question must now be occupied by their opponent, because either the cell already was marked before, or the tie break decided in the other player's favor.

The terminal condition for our blind version of Tic-Tac-Toe can be taken as is from the description of the standard game. Also the rules for the goals are the same as before but need to be extended to the possibility that both players complete a line in the same round.

```
terminal :- line(x)
terminal :- line(o)
terminal :- ~open

goal(white,100) :- line(x) & ~line(o)
goal(white, 50) :- line(x) & line(o)
goal(white, 50) :- ~open & ~line(x) & ~line(o)
goal(white,  0) :- ~line(x) & line(o)

goal(black,100) :- ~line(x) & line(o)
goal(black, 50) :- line(x) & line(o)
goal(black, 50) :- ~open & ~line(x) & ~line(o)
goal(black,  0) :- line(x) & ~line(o)
```

The supporting concepts `line(`*w*`)` and `open` are defined as before; see Chapter 2.

17.4 Card Games and Others

The two new keywords in GDL-II can be used to describe all kinds of card games, which are typically characterised by both randomness (shuffle) and information asymmetry (individual hands). For example, a single card dealt face down to a player can be specified thus.

```
legal(random,deal(Player,Card)) :-
   role(Player) &
   distinct(Player,random) &
   true(indeck(Card))

next(holds(Player,Card)) :-
   does(random,deal(Player,Card))
sees(Player,Card) :-
   does(random,deal(Player,Card))
```

Here, only the player who is dealt the card can see it. Multiple cards can be handed out in a single move that takes each card as a separate argument and of which players only get to see the argument position for their card.

In contrast, a card dealt face-up, say like in Texas hold'em, would be described as follows.

```
legal(random,deal(river(C))) :-
   true(indeck(C))

next(river(C)) :-
   does(random,deal(river(C)))
sees(P,river(C)) :-
   role(P) &
   distinct(P,random) &
   does(random,deal(river(C)))
```

With the last rule all players are informed about the river card.

Games involving communication among players, both public and private, can also be described in GDL-II. For example, as part of a negotiation a player `P` may offer a player `Q` to exchange an item

c for another item d. This can be formalized by the GDL-II clauses below.

```
legal(P,ask(Q,trade(C,D))) :-
   true(has(P,C)) &
   true(has(Q,D)) &
   distinct(P,Q) &
   distinct(C,D)

sees(Q,offer(P,C,D)) :-
   does(P,ask(Q,trade(C,D)))
```

Under these rules, their communication is private: only the addressee gets to see the offer.

17.5 GDL-II Game Management

GDL-II requires a modified protocol for running a game so that players are no longer automatically informed about everyone's moves after each round (cf. Chapter 3). Rather, each player gets his or her individual percept, or percepts, as determined by the game rules for the new keyword `sees`.

Shown below is the format of the Play message for GDL-II games in the current GGP communication language. The parameters are, (1) the usual match identifier; (2) an integer counting the number of turns; (3) the move that was executed by the player in the last turn; and (4) a list of the player's percepts.

(play *id turn move percept*)

If the player receives no information according to the game rules, the *percept* field is `nil`. The number of turns and the confirmation of the previous move help players recover from communication and other errors, when the game master selected a move on their behalf. On the first request, *turn* is `0` and both the *move* field and the *percept* field are `nil`.

The specification of the Stop message is likewise modified.

(stop *id turn move percept*)

Here is a sample of messages for a quick game of Blind Tic-Tac-Toe. As always, the Game Manager initiates the match by sending a `start` message to all of the players with their individual role, the rules of the game, and the values for the *startclock* and *playclock*. The players then respond with `ready`.

Game Manager to Player x: `(start m23 white ( ... ) 10 10)`
Game Manager to Player y: `(start m23 black ( ... ) 10 10)`
Player x to Game Manager: `ready`
Player y to Game Manager: `ready`

The manager starts play by sending an initial `play` message to all of the players. In this case, the first player responds with the action `(mark 1 1)` while the second player chooses `(mark 2 3)`.

Game Manager to Player x: `(play m23 0 nil nil)`
Game Manager to Player y: `(play m23 0 nil nil)`
Player x to Game Manager: `(mark 1 1)`
Player y to Game Manager: `(mark 2 3)`

The Game Manager checks that these actions are legal, chooses a legal move for random, updates the state of the game according to this joint move, and then sends `play` messages to the players to solicit their next actions. Since the players aimed at different cells, they both get their "`ok`". On the next step, the two players attempt to mark the same cell and both play `(mark 2 2)`.

Game Manager to Player x: `(play m23 1 (mark 1 1) (ok))`
Game Manager to Player y: `(play m23 1 (mark 2 3) (ok))`
Player x to Game Manager: `(mark 2 2)`
Player y to Game Manager: `(mark 2 2)`

Again, the Game Manager checks legality, randomly chooses a legal move for `random`, updates its state, and sends a `play` message requesting the players' next actions. In this case, the random move decided in favor of the first player, which therefore is the only player to again see "`ok`". The first player takes advantage of the situation and plays `(mark 3 1)` while the second player chooses `(mark 1 3)`.

Game Manager to Player x: `(play m23 2 (mark 2 2) (ok))`
Game Manager to Player y: `(play m23 2 (mark 2 2) nil)`
Player x to Game Manager: `(mark 3 3)`
Player y to Game Manager: `(mark 1 3)`

With this move, the game is over. The Manager lets the players know by sending a suitable `stop` message, stores the results in its database for future reference, and terminates.

Game Manager to Player x: `(stop m23 3 (mark 3 3) (ok))`
Game Manager to Player y: `(stop m23 3 (mark 1 3) (ok))`
Player x to Game Manager: `done`
Player y to Game Manager: `done`

17.6 Playing GDL-II Games: Hypothetical States

Most of the techniques that we have dealt with in this book can be applied to GDL-II unchanged: Factoring, the discovery of heuristics, and logical reasoning all operate on the elements of a game description. Their syntax and semantics is the same in both complete- and incomplete-information games.

But there are additional challenges specific to GDL-II. General game-playing systems need to be able also to draw the right conclusions from their partial observations. An example of this was mentioned above, when the absence of the observation "`ok`" implied that a particular cell had to be occupied by the opponent. This illustrates the kind of logical reasoning that GDL-II games require.

Players also need to evaluate the value of a move under incomplete knowledge of the state. They can do so, at least in principle, by computing a complete table of all possible positions after each round. This table is called an *information set* in mathematical game theory. It is constructed in a similar way to a game tree. Beginning with the initial position, which is fully known, a player can compute the resulting states for all combinations of legal moves. Later in the game, the player can generate every legal successor state for every possible current state. The percepts made after every round serve as filters. They allow to exclude any element from the information set that, according to the game rules, would have led to an observation different from the actual one.

Figure 17.1 shows an example of the possible states from white's perspective after the first round in Blind Tic-Tac-Toe.

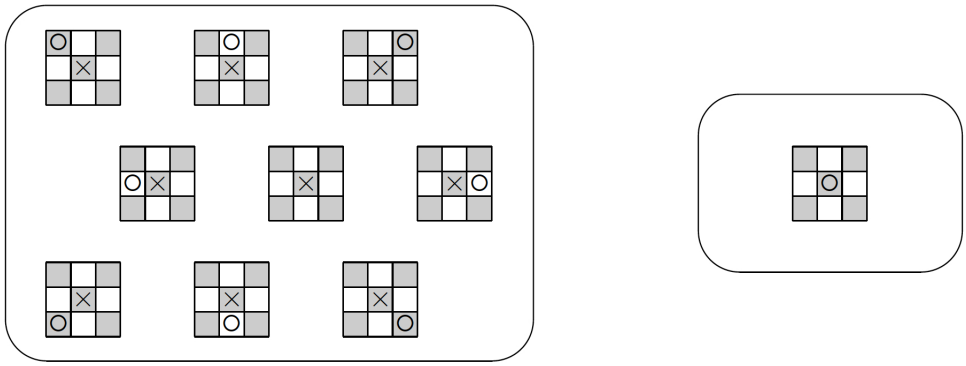

Figure 17.1 - The set of positions that white - the player playing crosses - considers possible after selecting the middle square in the first move and, respectively, seeing `ok` (left-hand side) or seeing nothing (right-hand side).

The depicted information sets illustrate how a general game-playing system can draw logical conclusions from a complete table of possible states. Square (2,2) carries an "`x`" in all elements of the first set. It follows that white's move must have been successful. On the other hand, (2,2) carries an "`o`" in each element of the second set, which in fact is a singleton. Again, white can deduce everything that follows logically from the absence of the expected percept. Black of course will likewise know that the cell in the middle has been marked in this case. Unlike white, however, black has no way of knowing that this is the only square that is occupied after the first round.

Provided the set of possible states remains sufficiently small throughout a game, a player can use this set to determine all moves that are guaranteed to be legal. The evaluation of a move can likewise be based on the entire information set. A player can, for instance, follow the minimax principle, for example, and always choose the move with the highest minimum value across all possible positions.

Blind Tic-Tac-Toe is an example of a game with a manageable table of possible states. The possible moves by the other players, including `random`, lead to further branching as the game progresses. On the other hand, the players gain information after each round, which helps to contain the growth of the information set.

But this is obviously not the case for larger imperfect-information games, like chess without revealing opponents' moves or where a deck of cards is shuffled at the beginning. Maintaining the complete information set is practically impossible in these games.

17.7 Sampling Complete States

A simple solution is to apply a technique similar to Monte Carlo tree search (cf. Chapter 8). Rather than generating all possible states, a player can restrict the computation of successor states to just a few, randomly selected joint moves. This allows to compute with only a few elements from the information set. Each of these hypothetical states will be tested against the player's percept after each round. If inconsistent, the hypothesis will be discarded or replaced by another randomly selected possible state.

Every ordinary GDL player can thus be lifted to GDL-II, since each element picked from the information is a complete state. A single hypothetical state can thus be treated by the player as if it was the actual state in a perfect-information game. Moreover, these hypothetical states can be analyzed independently. The individual results from this analysis can then be combined into an expect value for each move across the different possible states. The independence of the hypotheses allows the easy parallelization of this process.

But playing GDL-II games by randomly sampling complete states has its limitations. A player that considers only a small subset of all possible states may draw wrong conclusions. Even the mere legality of a move is not guaranteed by its being legal in just a few sample states. A move that

seems promising in some states may likewise turn out not to be a good move if all possible positions would be considered.

A further and more fundamental disadvantage lies in the implicit assumption of complete information when playing with individual elements of the information set. In so doing we completely ignore the difference between knowledge and the lack thereof. As a consequence, a move will never be considered useful if its sole purpose is to gain information. This is so because additional information has no value for any hypothetical state that assumes complete knowledge anyway.

Let's look at a very simple example of a single-player game that demonstrates how little the sampling technique values knowledge and how it will never choose an information-gathering move. Play commences with the random player choosing a red or blue wire to arm a bomb accordingly. The player, which shall be called `agent`, may then choose whether or not to ask which wire was used; asking carries a cost of 10 points to the final score. Finally, the agent must cut one of the wires to either disarm - or accidentally detonate - the bomb.

```
role(random)
role(agent)

action(noop)
action(ask)
action(arm(C)) :- color(C)
action(cut(C)) :- color(C)

color(red)
color(blue)

base(step(1))
base(step(N))  :- succ(M,N)
base(armed(C)) :- color(C)
base(score(S)) :- score(S)

succ(1,2)
succ(2,3)
succ(3,4)

score(90)
score(100)

init(step(1))

legal(random,arm(C)) :- color(C) & true(step(1))
legal(random,noop)   :- ~true(step(1))
legal(agent,noop)    :- true(step(1))
legal(agent,noop)    :- true(step(2))
legal(agent,ask)     :- true(step(2))
legal(agent,cut(C))  :- true(step(3)) & color(C)

sees(agent,C) :- does(agent,ask) & true(armed(C))

next(step(N))    :- true(step(M)) & succ(M,N)
next(score(90))  :- does(agent,ask)
next(score(100)) :- does(agent,noop)
next(score(S))   :- true(score(S))
next(armed(C))   :- does(random,arm(C))
next(armed(C))   :- true(armed(C))

terminal :- true(step(4))

explodes :-
  true(armed(C)) &
  does(agent,cut(C))

goal(agent,S) :- true(score(S)) & ~explodes
goal(agent,0) :- explodes
```

Shown below is the full game tree, including the outcomes for the agent at each terminal node.

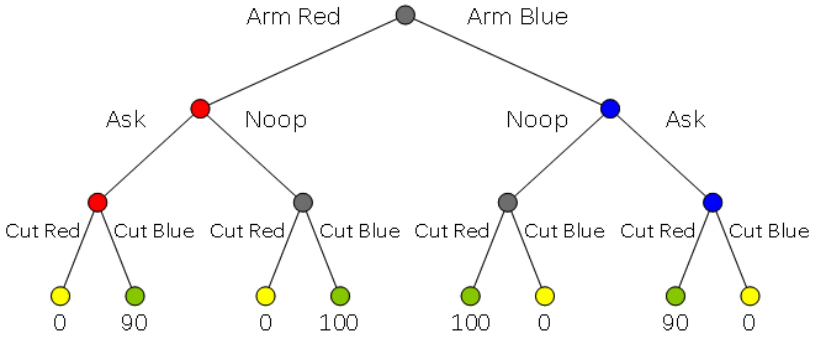

Figure 17.2 - The Exploding Bomb game tree

If your player samples the two states in the information set after round 1 and assumes complete knowledge in each of them, then in either case he believes he knows which color has been used to arm the bomb. Hence, he never asks the question in this game since it carries a penalty that he thinks he can avoid (due to superficial agreement of the samples). But after playing `noop` in round 2 the two samples (that is, the two gray nodes in Figure 17.2) do not agree on which move to take next, so that the player can do no better than choose randomly and thus obtain an average outcome of 50.

A variation of the Exploding Bomb game shows that a player also will never discount a move that gives away valuable information to opponents, for the same reason. In this version our player plays the arming agent - that chooses which wire arms the bomb - and also decides whether to tell its opponent which wire to cut. Freely providing this information carries a reward of 10 points.

```
role(agent)
role(opponent)

action(noop)
action(tell)
action(arm(C)) :- color(C)
action(cut(C)) :- color(C)

color(red)
color(blue)

base(step(1))
base(step(N))  :- succ(M,N)
base(armed(C)) :- color(C)
base(score(S)) :- score(S)

succ(1,2)
succ(2,3)
succ(3,4)

score(90)
score(100)

init(step(1))

legal(agent,arm(C))    :- color(C) & true(step(1))
legal(agent,noop)      :- true(step(2))
legal(agent,tell)      :- true(step(2))
legal(agent,noop)      :- true(step(3))
legal(opponent,noop)   :- ~true(step(3))
legal(opponent,cut(C)) :- color(C) & true(step(3))

sees(opponent,C) :- does(agent,tell) & true(armed(C))
```

```
next(step(N))    :- true(step(M)) & succ(M,N)
next(score(100)) :- does(agent,tell) & true(step(2))
next(score(90))  :- does(agent,noop) & true(step(2))
next(score(S))   :- true(score(S))
next(armed(C))   :- does(random,arm(C))
next(armed(C))   :- true(armed(C))

terminal :- true(step(4))

explodes :-
  true(armed(C)) &
  does(opponent,cut(C))

opposite_score(0,100)
opposite_score(10,90)
opposite_score(90,10)
opposite_score(100,0)

goal(agent,S)    :-  explodes & true(score(S))
goal(agent,T)    :- ~explodes & true(score(S)) & opposite_score(S,T)
goal(opponent,S) :- ~explodes & true(score(S))
goal(opponent,T) :-  explodes & true(score(S)) & opposite_score(S,T)
```

If our agent samples the information set after his first move and again makes the implicit assumption that both players play with complete information, then he believes that the opponent knows the right color anyway. Hence he always tells to avoid the penalty and thus is guaranteed to lose against a rational opponent (with a score of 10 to 90 points) while withholding the information would lead to a better outcome on average as the opponent has a mere 50/50 chance.

The two example games in this section are better approached by taking into account the entire information set of a player to decide on a move. But to reason correctly about one's own knowledge (and that of the other players) remains one of the many major challenges in general game playing for practical games where information sets are too large to be maintained explicitly.

Exercises

1. In Blind Tic-Tac-Toe, how many states are possible at the end of round 2 after you

    a. played `mark(2,2)` then `mark(1,3)` and your percepts were `nil` followed by `(ok)`?
    b. played `mark(2,2)` then `mark(1,3)` and your percepts were `nil` followed by `nil`?
    c. played `mark(2,2)` then `mark(1,3)` and your percepts were `(ok)` followed by `nil`?

2. Modify the rules of Blind Tic-Tac-Toe so as to describe a version without the `random` role, where

    - both players simultaneously choose a cell to mark;
    - they can choose any cell that is not already occupied with their own marker;
    - if they choose the same cell in the same move, then this cell remains blank;
    - otherwise, a player succeeds in marking a cell only if that cell was empty;
    - they see "`ok`" if and only if their attempt to mark a cell was successful.

3. Describe the following card game in GDL-II.

    A deck initially has 13 cards: ♣2, ♣3, …, ♣10, ♣J, ♣Q, ♣K, ♣A. Two players, `alice` and `bob`, are randomly dealt one card each from the deck. The first player then decides whether to bet or fold.

    - If she folds, the second player gets $5.
    - If she bets, the second player can decide to fold, check, or raise.
        - If he folds, neither player gets anything.
        - If he checks, the player with the higher card get $5.
        - If he raises, the first player can either fold or check.

- If she folds, the second player gets $5.
- If she checks, the player with the higher card wins $10.

The players get to see their opponent's card immediately after one of them has decided to check. If, however, the round ended with one of them folding, then the players are not informed about each other's hand. The game continues with two new cards dealt from reduced deck, but now `bob` has to make the initial choice between betting and folding choice. The game ends after six rounds, when there is only one card left in the deck. The player wins who has amassed the higher total amount.

4. Extend the game description from Exercise 3 by giving both players $100 at the start. When betting or raising, the players can decide how much they want to put in. When they check they have to match the previous bet (or raise, respectively), or go all-in. The minimum opening bet is always $5. The game ends prematurely when one of the players went broke.

5. Invent a simple (but non-trivial) game with imperfect information of your own. Describe in English. Write a GDL-II rulesheet for the game. Give a move history that takes the game from the initial state to a terminal state. Use the GDL stepper to show that your history works.

6. Implement the stochastic simulation of possible states as a bolt-on solution to your GDL player. Try your player out on one of the standard GDL-II games.