2013 IEEE/WIC/ACM International Conferences on Web Intelligence (WI) and Intelligent Agent Technology (IAT)

# A Poker Game Description Language

[*]João Castro Correia, [*]Luís Filipe Teófilo, Henrique Lopes Cardoso, Luís Paulo Reis
LIACC – Artificial Intelligence and Computer Science Lab
University of Porto
Porto, Portugal
[*]ei08128@fe.up.pt, [*]luis.teofilo@fe.up.pt, hlc@fe.up.pt, lpreis@dsi.uminho.pt

***Abstract*—During the last decade, Computer Poker has become the preferred test-bed for validating developments on the extensive-form game and multi-agent systems research domains. Because Poker is a game with hundreds of variants differing from each other by their betting structure, number of cards in the deck or winning conditions, numerous agents have been created for several different variants of the game. However, there is not a single unified description model that allows for those agents to be tested across different Poker variants inexpensively. For this reason, we introduce the Poker Game Description Language (PGDL), which, unlike other incomplete information GDL's, is uniquely focused on Poker agent development and testing. PGDL is integrated into a playable system which not only makes available a basic Agent Development API in Prolog, but also provides a simple in-built agent which can adapt to user-defined rules. In addition, this framework has a simple GUI which both basic and advanced test subjects demonstrated to be adequate and easy-to-use when defining new PGDL instances. We believe that despite the existence of more generic general game playing systems, the fact that our language natively supplies a shared infrastructure, common to all Poker variants, renders our approach very pertinent for Poker agent development. Tests demonstrated that our language was capable of describing the most popular Poker variants.**

***Keywords-computer poker; game description language; general game playing; poker variant; extensive-form games; incomplete information games***

## I. INTRODUCTION

In the past years, Poker has been an object of study and interest by AI researchers because it represents a completely different challenge from games such as chess. Games like chess deal with complete information, i.e., both players have full information about the current state of the game, making it possible to define a strategy through a decision tree. In contrast, Poker is a game of incomplete information where players only have information about their cards and the community cards, so it requires the construction of a probabilistic decision tree based on beliefs about the possible opponents' cards. Another challenge of Poker resides in its stochastic nature, i.e., there is the element of chance. This factor arises due to the fact that the cards are shuffled and randomly distributed.

Poker is also an industry with high growth rate that presents high profitability in the entertainment industry. It is played by millions of people around the world, both live and online [1]. It is a game with hundreds of variants, which differ from each other by betting structure, the number of cards in the deck, the way the winner is determined, among others.

However, to the best of our knowledge there is not a single unified description model that allows for game playing agents to be tested across different Poker variants inexpensively. For this reason, we introduce a new Game Description Language (GDL) for Poker games - Poker Game Description Language (PGDL), based on XML language. The goal of a GDL is to describe the state of a game as a series of facts and the game mechanics as series of logical rules. GDL's are typically used by General Game-Playing Systems (GGPS) as input. GGPS are systems that are capable of recognizing a formal description of a game and play the game effectively without human intervention. PGDL, unlike other incomplete information GDL's, is uniquely focused on Poker agent development and testing. Therefore, PGDL was developed to only identify the key concepts of Poker games rules in order to facilitate the definition of known or nonexistent Poker variants by users with Poker domain knowledge. To support the creation and assessment of PGDL entities, a general game playing system was also developed. This system allows users to play the PGDL described game against basic agents. The development of PGDL was divided in the following steps:

- Identification of Poker base rules with emphasis on the differences between its variants.
- Conceive a XML based language capable of specifying the identified rule differences.
- Construction of a system that recognizes the XML language (in Prolog) and that is capable of generating the specified game.
- Construction of a platform (PGDL Builder) that supports the creation of PGDL documents.
- Development of a generic agent that can play any variant described in PGDL.

The rest of the article is organized as followed: Section II describes the game of Poker with emphasis on the differences between its variants; Section III describes related work about general game playing systems and game

[*]Corresponding authors: João Castro Correia and Luís Filipe Teófilo
This work was supported by FCT – Fundação para a Ciência e a Tecnologia through the Scholarship with reference SFRH/BD/71598/2010.

978-1-4799-2902-3/13 $31.00 © 2013 IEEE
DOI 10.1109/WI-IAT.2013.131

description languages as well as Poker specific developments; Section IV describes the PGDL language with its composing entities detailed; Section V describes the system that was developed to support the creation of PGDL documents, with emphasis on implementation details; Section 6 describes the results of this paper by validating the system in-built agents, the graphical user interface and the creation of PGDL instances; Section 7 presents the articles main conclusions and pointers for future research.

## II. POKER

Poker is a card and betting game played by two or more players, without cooperation, i.e., each player plays for himself and against all others. Regardless of the played variant, the goal of Poker is always to win as much cash as possible and not to win a particular game. Due to its stochastic nature, it is impossible to mathematically ensure victory in a particular set of games. For this reason, a certain player is good when he or she manages to maximize profit when he or she is lucky and minimize prejudice when he or she is unlucky.

In Poker each player has to form a set of cards as valuable as possible – the hand of the player. Combinations that are less common are especially more valuable than regular combinations. Each player bets that his/her hand is stronger than the opponents' hands. Bets are placed in the pot and, in the end, the player with the strongest hand wins. However, if all players except one forfeit the game by folding, the last standing player wins the pot the game.

### A. Hand Ranking

A Poker hand is a set of five cards that expresses the player's score. Being $\Delta$ the set of all cards in the deck, $\Phi_i$ the set of private cards of a particular player $i$ and $\Omega$ the set of shared cards so that $\Phi_i \cap \Omega = \varnothing$ and $\Phi_i \cap \Phi_j$ for any players $i$ and $j$. Thus, the score function can be $score : [\Delta]^5 \rightarrow \mathbb{N}$. For a particular player $i$, the hand is the union of the pocket cards and the community cards ($\Phi_i \cup \Omega$). Thus, the player's score is given by the rank function, as follows:

$$Rank(\Phi_i, \Omega) = max(\{score(x) : x \in [\Phi_i \cup \Omega]^5 : \})$$

The possible hand scores are (from highest to the lowest score): Straight Flush, Four of a Kind, Full House, Flush, Straight, Three of a Kind, Two Pairs, One Pair and High Card. Examples of card combinations for each hand are presented on Table I.

TABLE I. POKER HAND RANKS WITH EXAMPLES

| Hand Name | | | | | |
|---|---|---|---|---|---|
| *Straight Flush* | 8♠ | 7♠ | 6♠ | 5♠ | 4♠ |
| *Four of a Kind* | A♣ | A♦ | A♥ | A♠ | K♠ |
| *Full House* | Q♣ | Q♠ | 7♥ | 7♠ | 7♦ |
| *Flush* | T♥ | 8♥ | 6♥ | 4♥ | 2♥ |
| *Straight* | 4♦ | 5♥ | 6♦ | 7♠ | 8♠ |
| *Three of a Kind* | T♣ | T♦ | T♥ | Q♣ | 3♦ |
| *Two pair* | 7♣ | 7♠ | 3♠ | 3♥ | Q♠ |
| *One pair* | 2♠ | 2♣ | 8♣ | 7♣ | 3♥ |
| *High Card* | A♥ | T♥ | 6♦ | 4♣ | 2♣ |

### B. Poker Variants

Poker is a group of similar games with the same base rule set. The denomination for a specific set of rules is called variant. The variants of Poker can be divided in 3 groups:

- Draw Poker – each player receives a set of private cards that only he/she can see and can improve the hand by card replacement. This group of games is usually played by casual players. Examples of Poker games that are part of this group are Five-Card Draw, Badugi and Kansas City Lowball;
- Stud Poker – each player receives a set of exposed cards (cards that belong to the player but everybody at the table can see) and a set of pocket cards that only the player can see, in multiple betting rounds. Six-Card Stud, Razz, Eight-or-better high-low stud are variations of Stud Poker;
- Community Card Poker – games in which each player receives a variable number of private cards to form an incomplete hand, which is completed by combining private cards with public shared cards (exposed to every player). The most popular poker variant nowadays, Texas Hold'em, belongs to this group as well as Omaha Hold'em and Manila.

Poker variants rules differ on the following features:

- Number of betting rounds – for instance, Texas Hold'em has 4 betting rounds and Five-card draw has 3 betting rounds.
- Number of private and public cards and the way they are dealt – in Texas Hold'em 5 public cards are dealt and each player receives 2 private cards, while in the Cincinnati 4 community cards are dealt, one before each round of betting, and each player has 4 private cards.
- Forced antes – some variants force all players to bet a certain quantity of money the ante before the cards are dealt.
- The betting order – there are variants such as Seven-card stud in which the first player to act is the one with the lowest exposed card and variants such as Omaha Hold'em where the first player is the one to the left of the big blind.
- The maximum number of players.
- Scoring – there are high-games in which the highest hand wins and low-games where the lowest hand wins. There are also high-low split games, where the best and the worst hands split the pot.
- Deck composition – there are variants that are played with only a few cards from the deck, such as Manilla (only cards above 7 with a total of 32 cards).
- Existence of wild cards – special cards that can score as any card (usually Jokers).
- Replacing cards – some variants, like Anaconda, allow players to pass cards between them in various ways. In other variants, like Badugi, players have the opportunity to improve their hand by discarding some cards and obtaining replacements from the dealer. There are also variants that force players to

discard a fixed number of cards, without replacement.

- Betting structure – Another major difference between the variants of poker is the betting structure. The structure can be limited, pot-limited and no-limit. The limit games are the ones in which there is a fixed value for each bet made by a player. In a pot-limited game no player can raise more than the size of the total pot. In these last two structures until winning the game there can be a limited number of raises during a round. In no-limit games there are no limits on bets.

Table II summarizes the main differences of the most popular and played Poker variants.

TABLE II. DIFFERENCES BETWEEN POKER VARIANTS.

| #Rounds | Cards | | | | | #Players |
|---|---|---|---|---|---|---|
| | | | | | *Wild* | |
| **Texas Hold'em** | | | | | | |
| 4 | 52 | Yes(5) | No | 2 | No | 2 to 9 |
| **Omaha Hold'em** | | | | | | |
| 4 | 52 | Yes(5) | No | 4 | No | 2 to 10 |
| **Baseball** | | | | | | |
| 4 | 52 | No | Yes(4) | 3 | 3/9 | 2 to 8 |
| **Cincinnati** | | | | | | |
| 5 | 52 | Yes(5) | No | 5 | No | 2 to 9 |
| **Five-card draw** | | | | | | |
| 2 | 52 | No | No | 5 | No | 2 to 6 |
| **Anaconda** | | | | | | |
| 4 | 52 | No | No | 7 | No | 2 to 7 |
| **Manilla** | | | | | | |
| 5 | 32 | Yes(5) | No | 2 | No | 2 to 9 |
| **Seven-card stud** | | | | | | |
| 6 | 52 | No | Yes(4) | 3 | No | 2 to 8 |

## III. RELATED WORK

Nowadays, it appears that information has become an increasingly valued resource as facilitator of decisions and processes of knowledge / intelligence in many different fields [2]. XML is currently a widely used language for representing information and will be used on this project to specify the rules of any poker variant. There are already some languages capable of representing concepts related to abstract games, such as the language of Zillions of Games platform, and to Poker games, such as HoldemML and PokerLang. The first two are also XML based languages.

### *A. General game playing systems*

A General Game Playing System is one that can accept a formal description of a game and play the game effectively without human intervention. General game playing is the design of artificial intelligence programs that can play more than one game successfully.

Zillions of Games platform is a popular and successful General Game Playing system for complete information games. Zillions is capable of playing almost every abstract board game, two-dimensional, or puzzles. In order to create a game, Zillions receives as input a file with the specification of rules written in a specific format, ZFR. After reading the ZFR file, the platform is able to generate the game and create intelligent and competent players that are able to play it. Zillions allows for the creation of multiplayer games, with either human and/or agents generated by the system [3].

ZFR language can represent most board games and puzzles by using S-expressions to define all components of each game [4]. ZFR represents concepts such as: the name of the game; a description of the game, i.e., a short explanation of the rules of the game, its history; the names of the players that will be identified in the game; the order in which the players play; the definition of the board; the definition of the game pieces and how they move on the game board; the initial game; the goal of the game.

### *B. Game description languages*

Game Description Languages describe the state of a game as a series of facts, and the game mechanics as logical rules. It's a set of high-level and rule-based formalisms used for communicating the rules of arbitrary games to general game-playing systems, whose challenging task is to learn to play previously unknown games without human intervention.

#### *1) Gala*

Gala is a system that allows the specification and efficient solution of large imperfect information games. The system takes a description of a game, analyzes it, and outputs strategies for the different players. The description of the game is written in a specification language, also called Gala [5].

#### *2) Ary*

Ary is a program that translates the rules of a game from the GDL into Prolog, and transmits them to a Prolog interpreter, that is used to generate legal moves, apply moves, determine when the game ends and determine the score for each player. This program won the 2009 and 2010 General Game Playing competition [6].

#### *3) GDL and GDL-II*

GDL is one of the most popular GDL's that is used to describe complete information games. GDL-II introduced new features to the language to also allow it to describe incomplete information games. Stochastic features were also introduced. This newest version GDL-II can be used to represent complex Poker games such as Texas Hold'em [7].

### *C. Poker-specific description languages*

Although the above described languages being generic enough to specify Poker games, they were not specially designed to do so. For that reason, developing Poker games in such languages and specially implementing new agents for them would be difficult given the amount of abstract concepts present on those languages. Our Poker specific GDL intends to solve this problem. There are no Poker GDL available (known to us) but the following Poker domain languages served as inspiration to our work.

#### *1) HoldemML*

HoldemML is a Poker agent development language [8] that consists of a generic framework for representing game logs for the Texas Hold'em variant. It is a XML based language that is used to summarize the events of the game, presenting advantages such as portability, interoperability,

and supporting tools. It represents concepts such as the players who have participated in the game, the cards each player has, the bets each player made at various betting stages and the game's winner.

*2) PokerLang*

PokerLang is a high level language of Poker concepts [9] designed to conceive high-level strategies for Texas Hold'em Poker agents. PokerLang represents concepts as strategies that players can use during the game, tactics that comprise the strategies of the players, conditions that activate a given strategy, including the number of players that are playing, the stack of the player or the player's position in the table. It also represents actions that comprise a tactic that can be set by the user or can be predefined through the Poker Builder, which allows users to create game strategies in an assisted and intuitive way, making this process more streamlined.

### *D. Other developments in Computer Poker domain*

First approaches to build Poker agents were rule-based, which involves specifying the action that should be taken for a given information set [10]. The next approaches were based on simulation techniques like in [11], i.e. generating random instances in order to obtain a statistical average and decide the action. These approaches led to the creation of agents that were able to defeat weak human opponents.

The great breakthrough in Computer Poker research was the discovery of the Counter Factual Regret Minimization Algorithm (CFR) in [12]. The CFR algorithm allows for the computation of a Nash Equilibrium strategy in large games like Poker through self-play. This could be done before through linear programming methods (like Simplex) but CFR is much faster because the processing time is proportional to the number of information sets instead of to the number of game states (about 6 orders of magnitude less). After the implementation of the original CFR, several variations of this algorithm emerged like CFR-BR [13].

## IV. PGDL Specification

In this section the structure of PGDL files is described. The PDGL format is based on XML. The format is enclosed in a hierarchical description of game rounds. The description of each game round compromises the flow of the game. There are also other elements to describe generic rules of the variant (such as the number of players) or meta-information (such as the name of the variant). Figure 1 summarizes the key components of the language by presenting the tree structure of a PGDL file.

### *A. Basic configuration*

The *PokerGame* is the root component of PGDL where it's detailed the name, the winning determination (High, Low or Mixed), the ante value and if the game is played with or without wild.

```
<PokerGame name="Leduc" wildCards="No"
   winningType="High" ante="1" />
```

Every *PokerGame* node must have a *Players* child node where the maximum and minimum number of players is defined.

```
<Players minimum="2" maximum="4" />
```

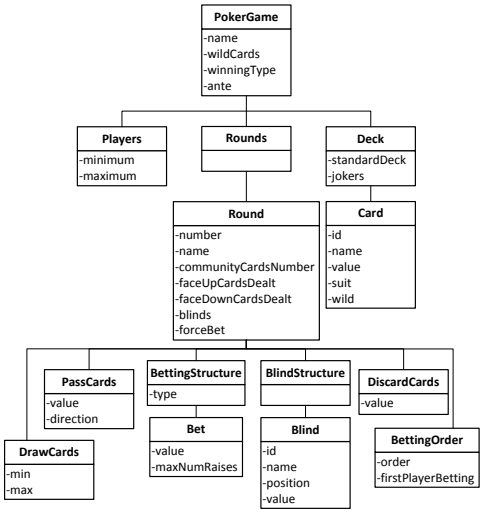


Figure 1. PGDL Specification

### *B. Deck personalization*

Poker games can be played with a standard deck (52 cards without Jokers) or with a partial deck with a given number of Jokers.

```
<Deck standardDeck="Yes" jokers="0" />
```

If the game is played with wild cards, any card can be used as wild (usually Jokers are used as the default wild card). Our deck definition allows not only using directly a standard deck but also personalize which cards belong to the deck, with custom names. This way one can even define Poker variants with two decks for instance. For each card one has to indicate the id and name of the card, the suit, it's value (any value of a standard card) and if that card is wild. This representation does not cover variants with dynamic wild cards.

Bellow the example of deck for Kuhn Poker (one of the simplest versions of Poker, used mainly for research purposes).

```
<Deck standardDeck="No" jokers="0">
   <Card id="k" name="King" value="K"
        suit="h" wild="No" />
   <Card id="q" name="Queen" value="Q"
        suit="h" wild="No" />
   <Card id="j" name="Jack" value="J"
        suit="h" wild="No" />
</Deck>
```

### *C. Round description*

The *Round* element is the most important component of the PGDL file structure because it is associated with the game flow. It describes how the rounds will take place during the game. Each round has a round number (to control the order of rounds), a name, the number of dealt shared cards, the number of faced up and faced down cards that

each player is dealt, one Boolean to control if the round must start with a bet and another one to if the round has blinds.

```
<Round number="1" name="Round One"
communityCardsNumber="1" faceUpCardsDealt="0"
faceDownCardsDealt="1" blinds="yes"
forceBet="no">
                    ...
</Round>
```

Furthermore, each round has sub-components: the Betting and Blind Structure of that round, the Cards Rules and the orders of the bets. Each round must have an individual betting structure defined.

The Betting Structure must be one of the three available types: Limit, No Limit and Pot Limit. Depending on the picked type, one has to indicate the maximum number of raises allowed per player and the bets' default value.

```
<BettingStructure type="noLimit">
   <Bet value="1" maxNumRaises="3" />
</BettingStructure>
```

*Blind Structure* only exists if the attribute *blinds* is activated (equals to 'yes'). This element contains a non-empty set of *Blind* elements. A *Blind* is described by a name, a unique id, the value of the blind and the position of the player that will post the blind.

```
<BlindStructure type="noLimit">
   <Blind id="smallBlind" value="1"
   name="Small Blind" position="nextDealer"
/>
</BlindStructure>
```

*Card Rules* are specified by three different elements: *Draw Cards*, *Discard Cards* and *Pass Cards*. *Draw Cards* indicates the minimum and maximum number of cards that each player can draw in a round. *Discard Cards* specifies the number of cards that each player must discard in that round. *Pass Cards* defines the number of cards that each player must pass and in which direction (clockwise or counterclockwise).

```
<DrawCards min="0" max="0" />
<PassCards value="1"
   direction="clockwise" />
<DiscardCards value="1" />
```

*Betting Order* it's a sub-component of the *Round*. To specify it, it's necessary to indicate in what order that round will occur (Clockwise or Counterclockwise) and i that the first player must play that round.

```
<BettingOrder order="clockwise"
   firstPlayerBetting="nextDealer" />
```

## V. PGDL System Implementation

The PGDL system is a set of applications that contemplate the following features:

- Support the creation of PGDL files through an intuitive GUI;
- Generate the user-defined Poker variants from a PGDL file or through the GUI;
- Allows the user to play the create Poker variant through a simple 2D game visualizer.

Figure 2 explains the workflow of the PGDL system. With PGDL Builder the user specifies the rules of a Poker game. That specification generates a PGDL XML Document that is validated by the PGDL XML Schema, to determine if the specification format is valid. After the validation is succeeded, the PGDL XML Document is then translated to Prolog file that contains the terms needed to configure a generic Poker implementation in Prolog. The Prolog implementation can be extended by a very simple Agent Development API. Two agents that used the agent development API are natively included: a Random Agent that picks a random action and a *E*[*HS*] Agent that plays based on the Expected Hand Strength of the current hand. After that, the game can be played in a 2D Visualizer by the user against the generated agents.

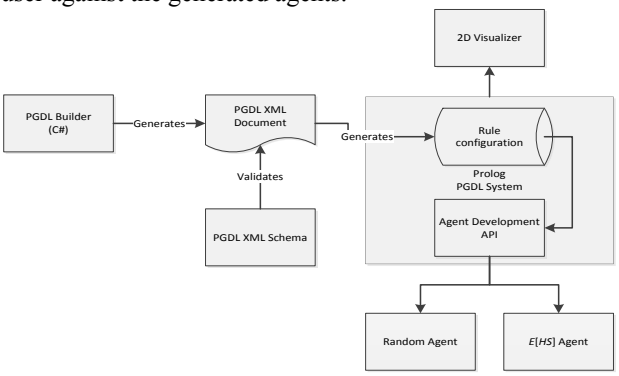


Figure 2. PGDL Builder System workflow

During the development of the PGDL system, several difficulties have emerged. In the following subsections those problems, their solution as well as implementation details will be depicted.

### A. Game rules configuration

The first problem to solve was to choose the best way to represent the list of terms in Prolog that specify the rules of a Poker variant. This set of terms was made to be accessible to either support the conversion of a PGDL files to Prolog and to be easily used by the generic Prolog system. Next we demonstrate an example of game rules configuration for the variant Leduc Hold'em (a simple variant mainly used for research purposes).

```
minPlayers(2).
maxPlayers(2).
stack(15).
name('Leduc').
winningType(high).
wildCards(0).
card(qs,'Queen of Spades',queen,spades,1,0).
card(js,'Jack of Spades',jack,spades,2,0).
card(ks,'King of Spades',king,spades,3,0).
card(qh,'Queen of Hearts',queen,hearts,4,0).
card(jh,'Jack of Hearts',jack,hearts,5,0).
card(kh,'King of Hearts',king,hearts,6,0).
round(1,1,1,0,1,'Pre Flop').
bettingStructure(1,noLimit,1,3).
```

```
blind(1,'Small Blind',1,leftDealer).
blind(1,'Big Blind',2,twoleftDealer).
bettingOrder(1,clockwise,leftDealer).
passCards(1,1,clockwise).
drawCards(1,1).
```

A round is a term that is composed of six atoms: number of round (order), the ante value, the number of faced up cards, the number of faced down cards, the number of shared cards and the name of the round.

BettingStructure is a term that has four atoms: the number of the round where it belongs, the type of betting structure, the value (that is only used when the structure is 'limit') and the maximum number of raises that are allowed in the corresponding round.

The term for card description is composed of an id, the name of the card, the value of the card, the suit, an auxiliary value and a binary value (1 or 0) that indicates if that card is wild or not.

### B. Representing a player state

During a game, the player is expressed as follows:

```
player(Id, Cards, PlayerType,
 PlayerAvailability, LastBet, Stack).
```

*Id* is a unique identifier for the player in the game. *Cards* is a list that contains the player's private cards. *PlayerType* indicates if a player is human or an agent (to allow it to be controlled by the GUI or not). *PlayerAvailability* indicates if that player is allowed to bet. The player will not be allowed to bet if it is in all-in mode or has forfeited the match. *LastBet* represents the total amount of cash that the player has betted during the current round (when a new round starts this value is set to 0. It is used to check if all player bets are matched). *Stack* represents the total amount of remaining chips of that player, in order to control the value of bets that the player can make.

### C. Representing the game state

The game state is represented by a list that contains a list of all players, the current value of the pot which is awarded to the winning player at the end of the game, the number of raises made so far (to be used in games that limit the number of raises), a list of shared cards and the position of the dealer. The last is used to locate the players in the table (relative positions to the dealer are used).

```
GameState = [NumberRaises-Pot-Dealer-
             SharedCards,PlayersList]
```

### D. Determining the end of a round

To determine if a round ended, the bet values of all available players are asserted to be the same as follows:

```
pass_aux(BetsList):-
 max_member(Max, BetsList),
 min_member(Min, BetsList),
 Max =:= Min.
```

When this happens, the round ends and the system moves to the next round. If there are no more rounds left, the winner of the game is determined.

### E. Determining the winner

Another problem faced was the way the winner is determined. To do this, the player with the best hand must by chosen. There are already lots of applications to compare Poker hands efficiently (descried in [14]) but, however, those are targeted to the most popular variants in which the hands are composed of at least 5 cards and a maximum of 7 cards. The fastest known evaluator is *TwoPlusTwo* Evaluator, which can evaluate about 15 millions of hands per second [14]. It takes a poker hand and maps it to a unique integer rank such that any hand of equal rank is a tie, and any hand of higher rank wins. *TwoPlusTwo* was used to calculate the winner in games that the hands are composed at least by 5 cards (for hands with more than 7 cards, we used the *TwoPlusTwo* 5 card lookup table and computed all combinations $C(n,5)$ of 5 cards to pick the best possible score). To compute the score of hands that are composed by maximum of 4 cards, a new evaluator was developed (since Straights and Flushes are not possible with less than 5). To do this, we assigned a value to each possible hand based on the cards that compose that hand. For example, if we have a hand of 4 cards ($C_1$, $C_2$, $C_3$, $C_4$) and the cards are all different the way the value of the hand is calculated is:

```
numEqualValue([C1,C2,C3,C4],HandValue):-
 max_member(Max,[C1,C2,C3,C4]),
 min_member(Min,[C1,C2,C3,C4]),
 delete([C1,C2,C3,C4],Max,L),
 delete(L,Min,L2),
 max_member(Max1,L2),
 min_member(Min1,L2),
 HandValue is Max*1000+Max1*100+Min1*10+Min.
```

### F. Dealing with wild cards

Another problem found was how to deal with wild cards when a player has in his hand wild cards and it's necessary to calculate the hand value. In that case the wild cards are identified and removed from the hand, creating a new hand. Then, the cards of the new hand are removed from the deck and with the new deck are generated all the possible combinations of the number of wild cards presented in the hand. Each one of those combinations are added to the hand and is calculated the value of that hand. The hand value is chosen from all the combinations of hands, according to the winning type of the game.

```
retrieveWildHandValue(Hand,WildCards,Value):-
 newHand(Hand,WildCards,NewHand),
 findall(C,card(C,_,_,_,_,_),Deck),
 newDeck(Deck,NewHand,NewDeck),
 length(WildCards,NumWC),
 length(L,NumWC),
 findall(L,comb2(NewDeck,L),AllCombs),
 getValue(NewHand,AllCombs,0,Value,_Card).
```

### G. Agent development API

An agent development API is included in the PGDL system. The agent development API supports information set abstraction features. The reason behind this is the fact that most Poker games usually have a very large decision tree

which makes it essential to abstract information sets (by making different cases undistinguishable) to enable agents to make decisions in reasonable time. There are three types of abstraction: moves sequence abstraction, information abstraction (card set abstraction in the case of Poker) and action abstraction (more useful for No Limit games with multiple possible raise amounts to choose from).

To implement an agent, one as to write the following Prolog terms:

- *abstract_hand(+Hand,-AbstractedHand)* – abstracts the hand of the player (private and shared cards). The default term is no abstraction (*abstract_hand(H,H)*).
- *abstract_history(+History,-AbstractedHistory)* – abstracts the sequence of game actions. Again, the default term is no abstraction.
- *play(+AbstractedHand,+AbstractedHistory, -AbstractedAction)* – the actual term that is used to play. It returns an abstracted action.
- *translate(+AbstractedAction, -Action)* – translates an abstracted action to an actual action to be executed by the agent.

The strategy of an agent is then defined as follows:

```
strategy(PID,SharedCards,History,Action):-
 player(PID, PCards, _,_,_,_),
 concat(PCards, SharedCards, Hand),
 abstract_hand(Hand,AbstractedHand),
 abstract_history(History,AbstractedHistory),
 play(AbstractedHand,AbstractedHistory,
      AbstractedAction),
 translate(AbstractedAction,Action).
```

## H. In-built agents

Two pre-built agents are included in the PGDL system: a random agent and a *E*[*HS*] (expected hand strength) based agent. The random agent picks a random action for any information set, avoiding folding (forfeit) when a check action (free pass) is possible.

The *E*[*HS*] agent is based on *E*[*HS*] equation. The Expected Hand Strength is the probability of the current hand of a given player being the best if the game reaches a showdown with all remaining players. For a player *i* against a giver number of opponents *n*, the *E*[*HS*] is given by:

$$E[HS]_n(i) = \left( \frac{Ahead(i) + \frac{Tied(i)}{2}}{Ahead(i) + Tied(i) + Behind(i)} \right)^n$$

The implemented agent uses the *E*[*HS*] value to choose the action according to Table III. For each betting structure, the agent as a fixed probability of following each action.

TABLE III. E[HS] AGENT STRATEGY

| E[HS] Value | Limit | | | No-Limit | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Fold | Call | Raise | Fold | Call | Raise 10% | Raise 20% | Raise 50% | All-In |
| < 30% | 100% | 0% | 0% | 100% | 0% | 0% | 0% | 0% | 0% |
| 30-50% | 50% | 30% | 20% | 50% | 30% | 10% | 3% | 2% | 0% |
| 50-80% | 5% | 50% | 45% | 5% | 50% | 25% | 10% | 5% | 5% |
| 80-100% | 1% | 19% | 80% | 1% | 19% | 20% | 15% | 15% | 30% |

## I. Graphical User Interface

In order to make it easier and more intuitive for a user to specify the rules of a poker game, a GUI was developed using Microsoft C# 4.0 Windows Forms. The interface was divided in four parts: Game, Rounds, Deck and Visualization. Four screenshots of each part are respectively presented on Figure 3.

The first screenshot presents the interface used to specify the Game's general rules. In it the user has to indicate the minimum and maximum number of players that can play the game, the way the winner is determined, the name of the game and if the game has dealer or not.

In the second screenshot is represented the interface used to define the rounds. The user has the possibility to choose the name of the round, the betting structure, the betting order, the rules that involve cards, and the blind structure where he or she can add the blinds that will occur in the game and the cards dealt. Each round is defined in different tabs. In each tab it is possible to edit that round. The order of the rounds is defined by the order of the tabs in the interface. The rounds can be re-ordered by drag & drop.

To specify the composition of the deck (third screenshot), the user has either the possibility of choosing to use the standard deck in a checkbox. If not, the user has to select each card one by one from the list on the right. The user must also indicate if the game has wild cards or not. If it has, he or she has to indicate how many jokers will be used or indicate if a particular card is wild or not.

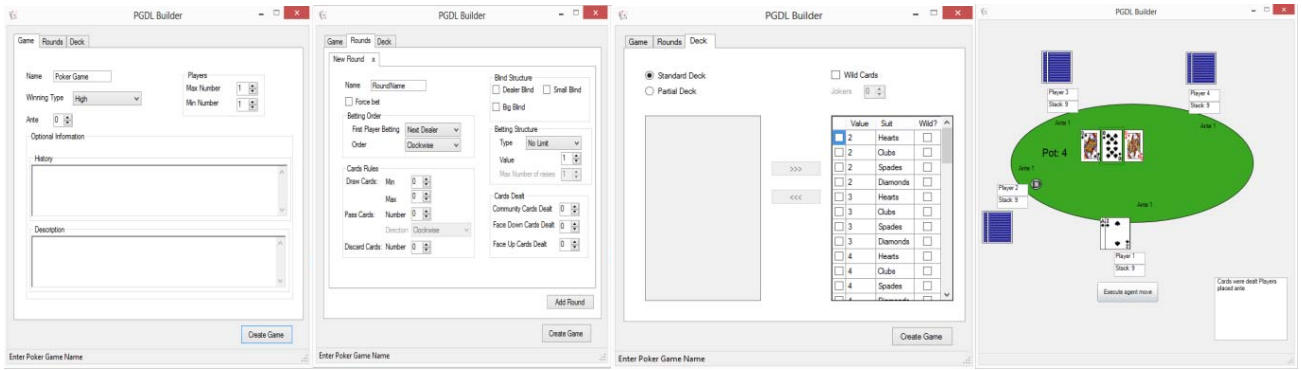


Figure 3. PGDL System GUI

To create the game the user has to click in the "Create Game" button. If the specification has errors the user will be notified. If not, the XML and Prolog file with the specification of the rules of the game will be created and the button to play the game in the 2D visualizer will be available. The 2D visualizer can be seen on the fourth screenshot in Figure 3.

## VI. Tests and Results

To validate the PGDL system, several tests were perfomed. First several popular Poker variants were implemented to confirm that the PGDL specification was sufficient to describe them all. Next, we tested the *E*[*HS*] agent against the random agent to assess if it is competent enough against the most possible basic agent – the random agent. Finally, we tested the GUI with several users to assess if the system is user-friendly to implement Poker variants.

The following Poker variants were implemented successfully with the PGDL specification: No-limit / Limit Texas Hold'em, Kuhn, Leduc, Cincinnati, Five-card draw, Anaconda, Manilla and Seven-card stud. In all implemented variants, the *E*[*HS*] agent was capable of beating the Random agent by a large margin (245.63 in milli-big-blinds/game in average).

To check if the GUI is user-friendly and intuitive, usability tests were performed. The test consisted of users (16 subjects on our tests) implementing two simple variants of poker, Kuhn Poker (2 times, one with standard deck and one with 3 card deck) and Leduc Hold'em Poker. All subjects were able to complete the task with an average time of 3.69 minutes. By analyzing the results of the tests we concluded that the time spent by the users doing the test was very similar, despite the level of knowledge about the Poker domain (standard deviation of 76 seconds). The learning curves of our tests can be observed on Figure 4.

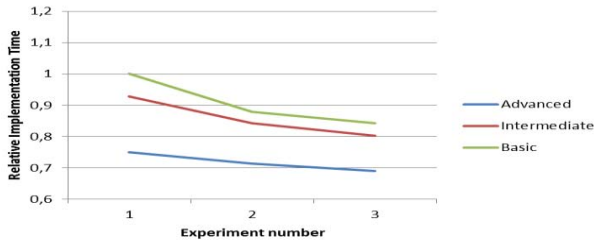


Figure 4. Learning curves using PGDL GUI

The biggest problems faced in the GUI usage were related to the understanding of the Poker specific nomenclature, even for users that said that they played Poker regularly. This is due to the fact of most Poker variants being unknown even for regular Poker players (the most played are Limit and No Limit versions of Texas Hold'em and Omaha Hold'em).

## VII. Conclusions

This research presented a generic system for creating poker variants. We created a XML dialect to represent the specification of most known Poker variant rules. From that specification, the developed system can generate a playable implementation of the game in Prolog. Excluding Omaha Hold'em (the system does not support that variant winning conditions), we were able to implement the most popular Poker variants with our system, proving its usefulness. The results of tests showed that the interface is user-friendly well designed and is easy to use because all the users took similar time to specify the same poker variant. This approach can enhance not only the easy implementation of any poker variant but also the creation of new variants. For future work, we will study the possibility of integrating more in-built intelligent agents that can compete with human players. For that, one could implement a more general version of the Counterfactual Regret Minimization algorithm in order to generate Nash Equilibrium strategies for the specified variant (which proved to be quite competitive in scientific agent competitions such as the ACPC – Annual Computer Poker Competition).

## References

[1] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," Artificial Intelligence, vol. 134, no. 1–2, pp. 201–240, 2002.

[2] C. H. Marcondes, "Representação e economia da informação," Ciência da Informação, vol. 30, no. 1, pp. 61–70, Apr. 2001.

[3] "Zillions of Games - Unlimited Board Games & Puzzles." [Online]. Available: http://www.zillions-of-games.com/.

[4] I. P. dos Reis and L. Reis, "Generic Interface for Developing Abstract Strategy Games," in 6th Iberian Conference on Information Systems and Technologies (CISTI), 2011, no. February 2011, pp. 950–953.

[5] D. Koller and A. Pfeffer, "Artificial Intelligence Representations and solutions for game-theoretic problems," Artificial Intelligence, vol. 94, pp. 167–215, 1997.

[6] M. Jean and T. Cazenave, "Ary , a general game playing program The Game Description Language," in Board Games Studies Colloquium, 2010, pp. 1–9.

[7] M. Thielscher, "A general game description language for incomplete information games," in Proceedings of AAAI, 2010, pp. 994–999.

[8] L. F. Teófilo and L. P. Reis, "HoldemML: A framework to generate No Limit Hold'em Poker agents from human player strategies," in 6th Iberian Conference on Information Systems and Technologies (CISTI 2011), 2011, pp. 755–760.

[9] L. Reis, P. Mendes, L. Teófilo, and H. Cardoso, "High-Level Language to Build Poker Agents," in Advances in Intelligent Systems and Computing Volume 206, 2013, no. July, pp. 643–654.

[10] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, "Opponent modeling in poker," in Proceedings Of The National Conference On Artificial Intelligence, 1998, vol. pp, pp. 493–499.

[11] D. Billings, D. Papp, L. Peña, J. Schaeffer, and D. Szafron, "Using Selective-Sampling Simulations in Poker," in AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information,, 1999, pp. 13–18.

[12] M. Zinkevich, M. Bowling, and N. Burch, "A new algorithm for generating equilibria in massive zero-sum games," in Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI), 2007, pp. 788–793.

[13] M. Johanson, N. Bard, N. Burch, and M. Bowling, "Finding Optimal Abstract Strategies in Extensive-Form Games," in Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12), 2012, pp. 1371–1379.

[14] L. F. Teófilo, L. P. Reis, and H. L. Cardoso, "Computing Card Probabilities in Texas Hold'em," in CISTI'2013 - 8a Conferência Ibérica de Sistemas e Tecnologias de Informação, 2013, pp. 989–994.