Computer Science and Information Systems 11(4):1249–1269 DOI:10.2298/CSIS130921029T

Rule based strategies for large extensive-form games: A specification language for No-Limit Texas Hold'em agents

Luís Filipe Teófilo[1,2], Luís Paulo Reis[1,3], Henrique Lopes Cardoso[1,2], Pedro Mendes[2]

[1] LIACC – Artificial Intelligence and Computer Science Lab.
R. Campo Alegre 1021 4169-007 Porto, Portugal
[2] FEUP – Faculty of Engineering, University of Porto – DEI
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
[3] EEUM – School of Engineering, University of Minho – DSI
Campus de Azurém 4800-058 Guimarães, Portugal
luis.teofilo@fe.up.pt, lpreis@dsi.uminho.pt, hlc@fe.up.pt

Abstract. Poker is used to measure progresses in extensive-form games research due to its unique characteristics: it is a game where playing agents have to deal with incomplete information and stochastic scenarios and a large number of decision points. The development of Poker agents has seen significant advances in one-on-one matches but there are still no consistent results in multiplayer and in games against human experts. In order to allow for experts to aid the improvement of the agents' performance, we have created a high-level strategy specification language. To support strategy definition, we have also developed an intuitive graphical tool. Additionally, we have also created a strategy inferring system, based on a dynamically weighted Euclidean distance. This approach was validated through the creation of simple agents and by successfully inferring strategies from 10 human players. The created agents were able to beat previously developed mid-level agents by a good profit margin.

Keywords: expert systems, knowledge representation, decision support systems, computer poker, rule based strategies, specification language

1. Introduction

Poker is probably the most popular card betting game in the world. It is played by millions around the world and has become a very profitable industry. Given its growing popularity and the amounts of money involved (billions of dollars), Poker also became a research subject in very different domains such as Mathematics, Artificial Intelligence or Sociology.

Poker's key features such as incomplete knowledge, risk management, need for opponent modeling and dealing with unreliable information, have turned this game into an important topic in Computer Science, especially for artificial intelligence. These features make it possible to use this game as an easy tool to measure progress in artificial intelligence research. This is so as to assess new approaches one only has to test them against the former ones – these tests can be easily performed using simulation tools.

Since the number of online players keeps on growing, several tools have been created to assist them during playing. Most of these tools are statistics-based applications that store information about played games, thus creating statistical knowledge about opponents. This can be used to help players making the decisions that are more lucrative at long term. Most of these systems classify the opponents' playing style – the extraction of this information is usually trivial since it is based on simple measurements like the absolute frequencies of opponents' actions, for instance. However, such systems neither suggest which action should be taken against those opponent profiles nor do they allow for configuring those suggestions with user-defined rules. The introduction of a recommendation system could enable potential users to customize their own strategy so as to collect suggestions about their preferred way to play. Thus, the user could consistently make better decisions regardless of external factors such as the devolution caused by fatigue. The creation of an agent that automates decisions can reduce this error even further, where no user interaction is required.

Besides the goal of assisting Poker players on making their own agents, these type of systems – expert systems – can be used as a basis for the creation of an agent capable of overcoming the best human players, a scientific goal that was not yet achieved in extensive-form games with the size of Poker: for the simplest version of Texas Hold'em Poker (Limited betting and 2 players) there are about 3.589×10^{13} possible decision points [1].

The goal of this work is to create a software agent that follows user-defined strategies in order to promote assisted-playing. The agent's strategy should be configurable through high-level instructions, by a strategy specification language. Those instructions should be prompted by a graphical user interface to allow for users without programming skills to define a customized Texas Hold'em Poker agent.

This work has been divided into the following goals:

- Creation of a language of concepts that includes key notions behind poker strategies and agent behavior – *PokerLang*;
- Build a graphical user interface for this language – *PokerBuilder* – which eases the creation of new *PokerLang* files;
- Automatic generation of *PokerLang* strategies from human player logs – this has the purpose of imitating good players' strategies, if enough data is available;
- Development of a Poker agent that follows the language specification;
- Evaluate the interface usability and the performance of the developed Poker agent.

The rest of the paper is organized as follows. Section 2 presents this work's background and the notation used throughout the paper. Section 3 presents recent methodologies to develop agents and to represent information in Poker. Section 4 presents the specification of *PokerLang*. Section 5 presents *PokerBuilder* – a graphical application built to aid the creation of *PokerLang* files. Section 6 describes the procedure we have used to create an inference system of *PokerLang* strategies, from past games. Section 7 describes the implementation of an agent that follows *PokerLang* strategies. Section 8 presents some experiments and results. Finally, Section 9 concludes and points directions for future research.

2. Definitions and Background

The goal of this work is to create a language that conceptualizes strategies for a well-known and popular extensive-form game – No Limit Texas Hold'em Poker. An extensive-form game is a generic representation of a sequential decision problem in form of a tree where each edge represents a decision and each node represents a sequence of performed actions (history). The history is hereinafter denoted by h considering that $h \in H$, being H the set of all possible game sequences according to the game's rules. Also consider h' a history-prefix where $h = h' \,||\, x$. Therefore, a game G can be represented as the following tuple:

$$G = \langle H, Z, N, A, a: H \to N, u: N \times Z \to \mathbb{Q} \rangle \,|\, Z \subset H \tag{1}$$

Z is a subset of H and represents the game's terminal nodes i.e. the nodes where the game ends. N represents the set of players in the game and A is the set of all possible actions.

An extensive-form game also requires the definition of three functions. Function a gives the set of all possible actions for a given node (or history) where for any particular node $z \in Z$ we have that $a(z) = \emptyset$ and for any particular node $h \in H \backslash Z$ we have that $a(h) \neq \emptyset$. Function p returns the acting player of any game sequence. Finally, function u returns the utility (or score) of a given player at a terminal node.

Next, we present the specific characteristics of a Poker game, with emphasis in the variant used in this work – No Limit Texas Hold'em Poker.

2.1. No-Limit Texas Hold'em

Poker is a class of card and betting games played by two or more players, without cooperation, i.e., each player plays for himself and against all others. Poker has innumerous sets of rules called variants; regardless of the played variant the goal of Poker is always to maximize utility in a sequence of games and not just to win a particular game. Due to its stochastic nature, it is impossible to mathematically ensure victory in a particular game. 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 – where being lucky means getting a good set of cards.

2.2. Scoring in Texas Hold'em Poker

At the beginning of a game G, each player $i \in N$ is given a set of two playing cards (private cards) which we will denote as $P_i \subset D$, where D is the deck – set of all playing cards (usually a regular 52 card deck without Jokers) – and $\forall i,j \in N$: $P_i \cap P_j = \emptyset$. The private cards P_i are only visible to player i and may never be unveiled to other players. At certain moments of the game, some shared cards are revealed – we will denote $S \subset D$ the set of shared cards and $S_r \subseteq S$ the set of visible shared cards at round $r \in \{preflop, flop, turn, river\}$, where $\forall i \in N$: $S_r \cap P_i = \emptyset$, for all r. The shared cards are always visible to all players and are used in combination with the private cards to determine a

particular player's score. For any No Limit Poker variant, $S_{preflop} \subset S_{flop} \subset S_{turn} \subset S_{river} = S$, and for the game rules considered in this paper we have: $|S_{preflop}| = 0$, $|S_{flop}| = 3$, $|S_{turn}| = 4$, $|S_{river}| = 5$.

In Poker, the score of a player i is given by the best $w \subset P_i \cup S$: $|w| = 5$ where $score(w)$ is maximized, being $score : [D]^5 \rightarrow \mathbb{N}^+$ a function that returns the score of a 5 card set. Therefore, for any remaining pair of players i and j, player i wins against player j if $\max_{w \subset [P_i \cup S]^5} score(w) \geq \max_{k \subset [P_j \cup S]^5} score(k)$.

The score of 5 card sets is divided in ranks (*High Card*, *Pair*, *Two Pairs*, *Three of a Kind*, *Straight*, *Flush*, *Full House*, *Four of a Kind* and *Straight Flush*), each of which is divided into several sub-ranks. The total number of sub-ranks is 7462, therefore $\forall w \in [D]^5, score(w) \in [0,7461]$.

2.3. Rules and utility

After dealing the cards, the game begins. The game is played in turns that are grouped in four Rounds (Pre-Flop, Flop, Turn and River). In each player's turn, he or she can choose one of the following actions, that may increase or not the pot value (prize):

- *Call* – match the highest bet. If no bets were made, this action is known as *Check*.
- *Raise* – increase the highest bet. If the players bets his/her entire stack, this action is known as *All-In*.
- *Fold* – forfeit the game and the pot.

A round ends when all players have bet the same amount (but each one must act at least once in that round). When the last round finishes, the player with the highest ranked set of cards wins the game and collects the pot. Alternatively, it is also possible to win the game by inducing opponents to fold by making bets that they are not willing to match. Thus, since players' cards (pocket cards) are hidden, it is possible to win the game with a lowered score hand. This particular feature of the game's rules makes it difficult to assess a player's decision.

Regardless of the winning situation, $\forall z \in Z : \sum_{i \in N} u(i,z) = 0$, making Poker a zero-sum game. However, usually in online Poker the game is not zero-sum due to the casino's profit margin $e \in [0,1]$. Considering $e \neq 0$, the real utility of player i in node z is usually given by $u(i,z) \times (1-e)$ if $u(i,z)$ is positive and $u(i,z)$ otherwise. In this paper we assume $e = 0$. In order to complete the definition of a Poker game, we define the new game tuple as specified in equation 2.

$$G_P = \left\langle \begin{array}{l} H, Z, N, A, P, S, a, p, u, \\ s : N \times H \rightarrow \mathbb{Q}_{\geq 0}, \\ b : N \times H \rightarrow \mathbb{Q}, \\ r : H \rightarrow 2^N, \\ c : H \rightarrow \mathbb{Q}_{\geq 0}, v : H \rightarrow S_r \end{array} \right| Z \subset H \wedge R = \{x : x \subset N\} \wedge S_r \subseteq S \qquad (2)$$

First, the sets *P* and *S* (described in section 2.2) were included and they respectively correspond to the private and community card sets ($\forall i : P_i \subset P$). Functions *s*, *b*, *c*, *v*, and *r* were added to the original definition of *G*. Function *s* denotes the amount of remaining cash and *b* the amount of cash betted by a particular player for a given history *h*, which means that $s(i,h)+b(i,h)$ for any *i* and *h* is the amount of cash of player *i* at the start of the game. Function *c* returns the value of the current maximum bet. Function *v* returns the visible shared cards for a given history. Finally, *r* is the function that determines the set of remaining players for a given history (it excludes the players that have folded). Given these functions, we can determine the utility of a player. The value of the pot in *h* is $\sum_{i}^{N} b(i,h)$. Given Texas Hold'em rules, player *i*'s utility in a terminal node *z* is:

$$u(i,z) \in \left\{ -b(i,z), \min\left(\sum_{i}^{N} b(i,z), b(i,z) \times |N| \right) \right\} \middle| \, i \in N \wedge z \in Z \tag{3}$$

Given these definitions we can also detail the *a* function, for which we consider that $A = \mathrm{Q}_{\geq 0}$. The No Limit variant of Texas Hold'em Poker is characterized for having no limits in bets – the players can raise up to their remaining money:

$$\forall h \in H : a(h) \in [(\min(s(p(h),h), c(h) - b(p(h),h)), s(p(h),h))] \cup \{0\} \tag{4}$$

where 0 corresponds to a *fold* action, the lower limit to a *call* and the higher limit to *all-in*. The lower and the upper limit might be equal, if the player doesn't have enough cash to call – in that case, the player goes all-in.

3. Related Work

The first successful approaches to create Poker agents consisted of hard-coded strategy definitions, which involves specifying the action that should be taken for a given information set [2]. An information set is the name of a decision point in Poker; contrarily to other games, a player in Poker does not have the full game state information. Poker information sets $I_{i,h} = \{h, P_i, v(h)\} \middle| I_{i,h} \in I$ are composed of the game's action sequence, the player's private cards and the visible community cards. Other features can be extrapolated from *h*. Following approaches were based on simulation techniques [3], i.e. generating random game instances in order to obtain a statistical average and decide the action. These approaches led to the creation of agents that empirically proved out to be capable of defeating weak human opponents.

One great breakthrough in the domain of Computer Poker and other extensive-form games research was the development of the Counter Factual Regret Minimization Algorithm (CFR) [4]. CFR allows for the computation of a Nash Equilibrium approximation strategy in large games such as Poker through self-play. This could be done before through linear programming methods (e.g. Simplex), but CFR is much faster since the processing time is proportional to the number of information sets instead of the number of game states (about 6 orders of magnitude less). Several approaches

based on CFR, like Restricted Nash Response [5] and Data-biased response [6] backed up the first victories against Poker experts [7]. The main problem about CFR is that it is only proved to compute a Nash-Equilibrium approximation for two players. However, the strategies generated for more than two players still proved to be robust empirically. Another problem is that these types of strategies are fixed which means that they are unable to dynamically adapt to the changing game conditions.

Other recent methodologies based on pattern matching [8] and cased based reasoning [9] applied to Poker inspired this work, namely the *PokerLang* strategy inferring system described on Section 6. These approaches generate Poker agents based on past games played by human experts. As stated before, the number of possible decision points in Poker is enormous. For that reason, these approaches based their strategies on the concept of information set similarity. In [9], two information sets have a degree of similarity equal to the average similarity of the game features. In [8], instead of the average, the degree of similarity was calculated through the Euclidean distance between sets of features. Being *i* and *j* two information sets, $f \in F$ a game feature and i_f, j_f the values of feature *f* on those information sets, the distance is given by:

$$euclidean(i, j) := \sqrt{\sum_{f}^{F} (i_f - j_f)^2} \tag{5}$$

The Monte Carlo Search Tree algorithm [10] and reinforcement learning approaches [11] are other techniques that have been successfully applied to the domain of Computer Poker. A more throughout description of the most recent works can be found in the reviews [12, 13].

The approach followed in this work consists of defining the agent's strategy through a high level specification language. One example of a similar work is the Poker Programming Language (PPL) [14]. The main issue about PPL, however, is that it only considers low level features of Poker, which means that it takes a long time to specify a complete strategy. Moreover, the absence of advanced game concepts makes it only possible to create very basic and static strategies which can be easily beaten by an average opponent.

4. PokerLang

Due to its stochastic nature, Poker players use rather different strategies with similar game conditions. A strategy is used under certain information sets that are described by specific features $f \in F$ (being *F* the set of game features that can influence a decision at a certain point of the game) such as the card probabilities (hand strength), player's cash, number of opponents, playing order, among others. We refer to these features as the game's features – characteristics of the information set that influence player decisions.

A strategy *T* can be conceptualized as a set of tactics. A tactic $t \in T$ is a mapping between a set of information sets and a set of actions:

$$t : I' \rightarrow A' \mid I' \subset I \wedge A' \subset A \tag{6}$$

I' and *A'* represent two types of game abstraction: information set abstraction and action abstraction (respectively). This is done by transforming *F* into *F'*, where the features of *F'* are simplified so that $|I'| < |I|$. The information set abstraction is particularly essential because Poker has so many information sets that it would not be possible, with current hardware, to store the corresponding action for each one. For a similar reason, action abstraction is also handy; in No Limit Poker there is a continuous interval of possible decisions (see Equation 4)). Usually this interval is discretized into a fixed number of possible decisions: fold, call, intervals of raise values and all-in (betting the remaining cash). Using a fixed number of decisions facilitates search-tree strategy based algorithms, because it greatly reduces the horizontal and vertical expansion of the decision tree by reducing its branching factor.

In order to specify these concepts, we have created a high-level language– *PokerLang* – whose syntax and grammar was based on Coach Unilang [15]. Coach Unilang was successfully used in the robotics soccer domain. The generic approach of this language allows for its easy adaptation to other domains.

The language root starts by defining the concept of strategy: a strategy is a set of tactics each of which is a tuple composed by an activation condition and a behavior for that tactic. The activation condition consists of abstracting decision points or information to define *I'*. They correspond to a set of verifications of the visible game features (through evaluators) or predictions about uncertain events (through predictors). A tactic's behavior is the procedure followed by the player when the activation condition is met (the behavior itself has a second layer of verifications that can abstract the information set even further). The tactic's behavior could be either user-defined or language predefined (based on common expert tactics). In the next sub-sections we describe *PokerLang*'s main language concepts. Below we present the main elements of the language in BNF notation.

<STRATEGY>::= {<TACTIC>}
<TACTIC>::= <ACTIVATION_CONDITION> <TACTIC_BEHAVIOUR>
<ACTIVATION_CONDITION>::= {<EVALUATOR>}
<TACTIC_BEHAVIOUR>::= <PREDEFINED_BEHAVIOUR>|<BEHAVIOUR>
<PREDEFINED_BEHAVIOUR>::= loose_agressive|loose_passive|

tight_agressive|tight_passive
<BEHAVIOUR>::= {<RULE>}
<RULE>::= {<EVALUATOR> | <PREDICTOR>} <ACTION> <VALUE>
<ACTION>::={<PREDEFINED_ACTION><PERC>|

<DEFINED_ACTION><PERC>}

4.1. Evaluators

Evaluators are comparators of the game's visible features with fixed values. They are assertions that must be verified to activate the behavior of a tactic or a rule.

<EVALUATOR>::=<NUMBER_OF_PLAYERS> | <STACK> | <POT_ODDS>| <HAND_STRENGTH> | <HAND_REGION> | <POSITION_AT_TABLE>

Number of Players. This evaluator considers how many players one is competing against. The number of players is an important measure because the higher it is, the lower is the probability of success of any given hand. The number of players for the current history *h* is simply given by *r*(*h*), the number of remaining players.~

Stack. The stack is the relative amount of chips that a player currently has. The value has to be relative since there is a plethora of possibilities of a player's amount of chips. We consider the amount relative to the antes – mandatory bets made before the game starts. Considering h_0 the initial history (where the players already bet their antes), the relative value is given by:

$$M(h \in H, i \in N) := \frac{s(i,h)}{\sum_{j}^{N} b(j,h_0)} \tag{7}$$

The values of function *M* were discretized in this evaluator into five different zones (see Table 1). One can also use completely custom intervals, with the stack values always computed by the *M* function. Check the BNF code bellow for details.

<STACK>::= <PREDEFINED_STACK_REGION> | <STACK_REGION_DEFINITION>
<PREDEFINED_STACK_REGION>::= green_zone | yellow_zone | orange_zone | red_zone | dead_zone
<STACK_REGION_DEFINITION>::= <STACK_REGION_NAME>

<STACK_INTERVAL>
<STACK_REGION_NAME>::= [string]
<STACK_INTERVAL>::= <MIN_STACK> <COMP> <STACK_VALUE> <COMP> <MAX_STACK>
<MIN_STACK>::= <STACK_VALUE>
<MAX_STACK>::= <STACK_VALUE>

Table 1. User defined Stack Regions

Name	Stack/M
Green Zone	M >20
Yellow Zone	10 < M ≤ 20
Orange Zone	5 < M ≤ 10
Red Zone	1 < M ≤ 5
Dead Zone	M ≤ 1

Pot Odds. Pot Odds is the ratio between the size of the pot and the cost to call the maximum bet. Pot odds are usually compared with the hand's winning probability. When the pot odds are higher than the hand odds, the player should call (Equation 8).

$$Odds(h \in H, i \in N) := \frac{c(h) - b(i,h)}{\sum_{j}^{N} b(j,h)} \tag{8}$$

Hand Region. The probability of winning a game in Poker depends on the player's starting cards $P_i \in P$. There are $|P| = 1326$ possible combinations of starting hands. This poses a problem because if the user were to define a single tactic for every starting hand, the number of possible combinations would be enormous. To solve this problem, *PokerLang* uses bucketing – an abstraction technique that consists of grouping different hands that should be played in a similar way [6]. *PokerLang* allows users to either create their own buckets (HAND_REGION_DEFITION) or use Dan Harrington's (see Table 2) ones [16].

```
<HAND_REGION>::= <PREDEFINED_HAND_REGION> |
        <HAND_REGION_DEFINITION>
<PREDEFINED_HAND_REGION>::= a | b | c | d | e
<HAND_REGION_DEFINITION>::= <HAND_REGION_NAME> {<HAND>}
<HAND_REGION_NAME>::= [string]
```

Table 2. Dan Harrington's Groups

Group	Hands
A	AA, KK, AKs
B	QQ, AK, JJ, TT
C	AQs, 99, AQ, 88, AJs
D	77, KQs, 66, ATs, 55, AJ
E	KQ, 44, KJs, 33, 22, AT, QJs

Hand Strength. This evaluator is activated when the hand strength has a certain minimum value. The hand strength is given by the ratio between the number of hands that have lower score than the player's hand and the total number of possible hands [17]. It calculates, in node *h*, the probability of winning if the game reaches a terminal node $z \in Z$ where $r(h) = r(z)$, that is, considering that all current players reach the terminal node. The hand strength is given by the *HS* function (Equation 9). The HS_{oper} (equation 10) is an auxiliary function where <*oper*> is an arithmetical comparator (=, > or <).

$$HS(P_i, S, h) := \left(\frac{HS_{>}(P_i, S, h) + 0.5 \times HS_{=}(P_i, S, h)}{HS_{>}(P_i, S, h) + HS_{=}(P_i, S, h) + HS_{<}(P_i, S, h)} \right)^{|r(h)|} \tag{9}$$

$$HS(P_i,S,h)_{<oper>} := \left| \begin{Bmatrix} x : x \in [D \setminus P_i \setminus S]^2 \wedge \\ \max_{w \in [P_i \cup S]^5} score(w) < oper > \max_{k \in [x \cup S]^5} score(k) \end{Bmatrix} \right| \quad (10)$$

Position at table. The position at table is the player's relative position to the current Big Blind position. The later the position is the better chance the player has to observe his or her opponents' moves and act accordingly. Since games have a variable number of players, in order to better abstract the strategies, the position value is defined through the position quality $PQ \in \mathbb{N}^+$ which also depends on the type of the opponents:

$$PQ(h,i) := \left(\left| \{h' : h' \subset h \wedge p(h') = i\} \right| \bmod |N| \right) - \left| \{i : i \in N \wedge aggressive(i)\} \right| + \left| \{i : i \in N \wedge passive(i)\} \right| \quad (11)$$

Functions *aggressive* and *passive* assert if the player is respectively an aggressive or passive player. A player is aggressive if in past games (a collection of G_P values), the ratio between the number of raise and call actions is above 1, otherwise the player is passive. The range of possible position qualities depends on the number of players in the following proportion: [-(|*N*|-2), (|*N*|-2)]. For instance, in a 10 players table, the range would be [-8, 8].

<POSITION_AT_TABLE>::= <PREDEFINED_POSITION_REGION> |

<POSITION_REGION_DEFINITION>
<PREDEFINED_POSITION_REGION>::= bad_position |

normal_position | good_position
<POSITION_REGION_DEFINITION>::=
<POSITION_REGION_NAME>{POSITION}
<POSITION_REGION_NAME>::= [string]
<POSITION>::= <MIN_POS><COMP><POS_VALUE><COMP><MAX_POS>
<POS_VALUE>::= <INTEGER>

There are 3 predefined regions: *bad*, *normal* and *good*, respectively equations 13, 14 and 15 (equation 12 is auxiliary). The user is also allowed to define his/her own custom regions (see POSITION_REGION_DEFINITION in the code above).

$$\begin{cases} Min = -|N| + 2 \\ Max = |N| - 2 \\ TR = (|N| - 2) \times 2 \end{cases} \quad (12)$$

$$bad \in \left[Min, Min + \frac{TR}{3} \right] \quad (13)$$

$$normal \in \left]Min + \frac{TR}{3}, Max - \frac{TR}{3}\right[\tag{14}$$

$$good \in \left]Max - \frac{TR}{3}, Max\right] \tag{15}$$

4.2. Predictors

Predictors represent estimated game features. Since hidden information in Poker (opponents' cards) is crucial to the game's outcome, to be competitive a player must make predictions about what is the actual game state. Predictions are based on the opponents' moves on previous games.

<PREDICTOR>::= <IMPLIED_ODDS> | <OPPONENT_HAND> |
<TYPE_OPPONENT > | <STEAL_BET>
|<IMAGE_AT_TABLE>

Opponent Hand. This predictor estimates the possible opponent hand taking into account the player's cards and the community cards. For instance, if the opponent hand predictor is "Flush", this should be read as "If the opponent is able to reach a flush".

Steal Bet. The steal bet is the amount of chips you need to get the pot with a low score hand. It depends on the type of opponents that one is facing.

Implied Odds. This predictor corresponds to the pot odds but it takes into account the evolution of the player's hand. Let H^+ be the set of all possible subsequent histories to *h*, the implied odds can be calculated through equation 16.

$$implied\_odds(i,h) := \left| \frac{\sum_{j}^{N} b(j,h) + \frac{\sum_{h^+}^{H^+}\sum_{j}^{N} b(j,h^+)}{|H^+|}}{c(h) + \frac{\sum_{h^+}^{H^+} c(h^+)}{|H^+|}} \right| h \in H \setminus Z \tag{16}$$

Type of Player. This predictor considers the type of the last playing opponent in the table taking into account his/her past behavior in the game. There are 4 predefined types of opponents [18]: loose-aggressive, loose-passive, tight-aggressive and tight-passive.

<TYPE_OPPONENT>::= loose_agressive | loose_passive |

tight_agressive | tight_passive

4.3. Actions

As stated before, there are several possible values of actions in a Poker game. In *PokerLang*, the user can choose predefined moves (based on common expert moves) or custom moves. The predefined actions are usually a sequence of actions – they abstract decision points because when such action is activated, the action may control de agent's behavior throughout the rest of the round or even the rest of the game.

<ACTION>::= {<PREDEFINED_ACTION><PERC> |
<DEFINED_ACTION><PERC>}
<PREDEFINED_ACTION>::= <STEAL_THE_POT> | <SEMI_BLUFF> |
<CHECK_RAISE_BLUFF> | <SQUEEZE_PLAY> |
<CHECK_CALL_TRAP> | <CHECK_RAISE_TRAP>|<POST_OAK_BLUFF>

Moves can be customized by defining the distribution of bet amounts if the activation condition is met. The specified bet amounts are always relative to the current pot value in $h:\sum_{i}^{N} b(i,h)$. The distribution can be defined for the three rounds of the game so actions can be reused. The action can be a fold (BET_VALUE = 0) or a raise (BET_VALUE > 0). If the action is impossible to perform or not specified, the agent calls by default.

<DEFINED_ACTION>::= <ACTION_NAME>{<PRE_FLOP_ACTION> |
<FLOP_ACTION> | <TURN_ACTION> |
<RIVER_ACTION>}
<PRE_FLOP_ACTION>::= {<BET_VALUE><PROBABILITY>}
<FLOP_ACTION>::= {<BET_VALUE><PROBABILITY>}
<TURN_ACTION>::={<BET_VALUE><PROBABILITY>}
<RIVER_ACTION>::={<BET_VALUE><PROBABILITY>}

5. PokerBuilder

After defining the high-level language, the next phase of this work was to build a simple graphical application which allows for users to easily define new *PokerLang* strategies. *PokerBuilder* is an Adobe Flex application that allows users to define the strategy's rules using the concepts of the language previously introduced, and set the behavior of a poker agent. With a smooth interface and simple features, *PokerBuilder* is accessible to any user that understands the main concepts of poker. One of the purposes of this work was to make a very practical application, even usable for users only familiarized with the most basic computer usage.

For the implementation of the language of concepts, *PokerBuilder* is divided in four major classes: Strategy, Tactic, Rule and Property (Fig. 1). The interface begins with an instance of the Strategy Class that creates instances of all other classes depending on what the user is creating. *PokerBuilder* gives the user two different views to create rules: Strategy View and Tactic View.

The figures on the appendix section demonstrate this software's graphical user interface. The software includes a strategy builder (Fig. 5) to permit the user to create sets of tactics. To create a rule, the user must select its name, various evaluators and/or predictors and the corresponding action. Fig. 6 indicates an example of the interface used to select the evaluators, predictors and actions. Finally, if the user wants to select a pre-defined action (see section 4.3); he or she can use the interface shown in Fig. 7 to personalize them.

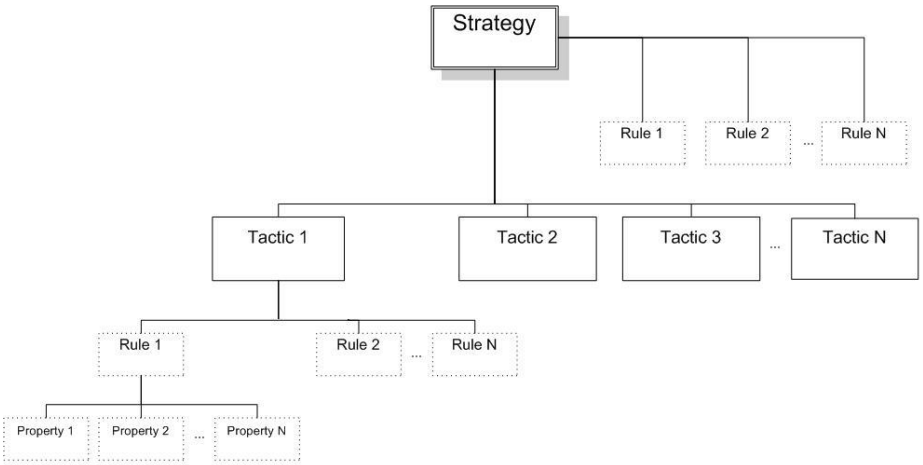

Fig. 1. PokerBuilder application schema

6. Inferring strategies from game logs

Although the *PokerBuilder* interface is easy to use, it still takes a long time to accurately describe a strategy with precision to achieve a good performance. In order to surpass this problem, we have designed an approach to perform inference of rules from game logs – sets of recorded games G_P. This way, if the user has available data from games of agents or humans that play with a strategy similar to the intended one, the user can just import those games, infer rules from them and then simply adjust the rules with *PokerBuilder*.

The built inferring system does not consider predictors; it just considers the following evaluators:

- Stack: *St*
- Hand Strength interval from $HS(P_i,S,h)$: *Hi*
- Position at table: *Po*

To build this system, we considered all possible combinations of the stated evaluators. However, since the hand strength is a continuous measure, its distribution has to be discretized. Let us analyze a distribution of hand strength values extracted from a particular collection of game logs, provided by Pedro Reis (see Fig. 2).

As expected, the frequency of high values of hand strength is higher on later rounds (right hand side of Fig. 2). This happens because the players successively give up weaker hands. Since the distributions are rather distinct, we differentiate them during the inferring process: when inferring evaluators on Pre-Flop rounds we use the distribution on the left, and for other rounds we use the distribution on the right.

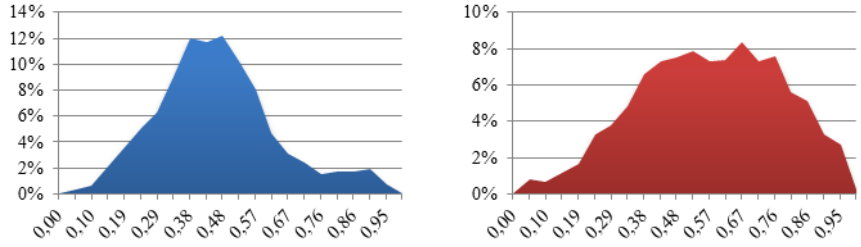

Fig. 2. Hand strength relative distribution observed from the dataset. On the left, the distribution on the Pre-Flop round and on the right the distribution on the Post-Flop round. The horizontal axis contains the values of hand strength (ranging from 0 to 1) and the vertical axis is the relative frequency of that hand strength value.

The discretization process was simple: a fixed number of hand strength intervals (k). The interval offsets were chosen to obtain a uniform distribution based on the relative frequency of $HS(P_i,S,h)$ values. A similar strategy was considered for the selection of actions Ad. The betting distribution was also obtained from the game logs collections (Fig. 3). After that, from the betting distribution a fixed number of intervals were extracted (q). Given this, the tuple that the inferring system must recognize is:

$$\langle St, Hi, Po, Ad \rangle \begin{cases} St \in \{green, yellow, orange, red, dead\} \\ Po \in \{bad, normal, good\} \\ |Hi| = k, |Ad| = q \end{cases} \quad (17)$$

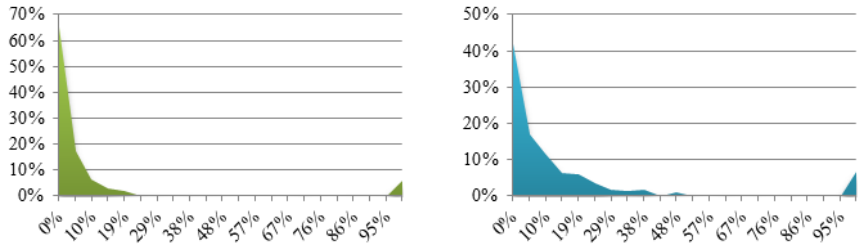

Fig. 3. Betting distributions. On the left the distribution for Pre-Flop and on the right the distribution for Post-Flop rounds. The horizontal axis expresses the percentage of the player's money that was betted.

The number of recognizable tuples is $|St| \times |Hi| \times |Po| \times |Ad| = 5 \times k \times 3 \times q$. In the experiments we arbitrarily used $k = 10$ and $q = 10$, making a total number of 1500 cases.

We used three different strategies to recognize a case from the game logs. First we used a well-known classifier – the Random Forest Tree – that already proved

empirically to be the best suited for Poker data [8]. The second strategy was to use the Euclidian distance (Equation 5) between the extracted features and features from the static tuples – the closest case is the one to be activated. Finally, we used a strategy based on the weighted Euclidean distance. The weighted Euclidian (Equation 18) distance considers a weight vector w where w_f is the weight of feature *f*.

$$weidist(s,d) := \sqrt{\sum_{f}^{F} w_f \times (s_f - d_f)^2} \tag{18}$$

The weight vector is determined empirically through the inferring system validation method. The validation method consists of creating an agent that follows the inferred strategy and then determining its accuracy when following the learned strategy:

$$acc(i,C,h,i^a) \frac{\left|\left\{1 : G_P \in C \wedge h \in G_P \wedge i \in G_P \wedge p(h) = i \wedge h_i^+ = h_i^+ a\right\}\right|}{|C|} \tag{19}$$

where *C* is the collection of cases for player *i*, h_i^+ is the history after the player *i* action and $h_i^+ a$ is the action performed by the agent representing player *i*. The accuracy is the ratio between the number of cases where the agent selected an action similar to the player's original action and the total number of cases.

In our experiments, to determine the weight vector, we generate its weights randomly so that $\sum_{i}^{|F|} w_i = 1$, generate the agent and then determine its accuracy for a fixed number of iterations. The agent with better accuracy is the one that it is selected by the system. Other policies can be used to determine the weights, namely genetic algorithms with populations of agents with different weight vectors. However, it is possible to check (Table 3) that the random generation policy already produced agents with very good accuracies. The weighted Euclidian distance always produced agents with greater accuracy than the two other methods, with an average accuracy of ~79% for datasets with 5000 cases and 10.000 iterations, proving the usefulness of this method.

Table 3. *PokerLang* strategy inferring accuracy. Logs of 10 different players. For each player, 3 sets of cases with different sizes were extracted (1000, 2500 and 5000). The game logs con-tained full game state description of the players from whom the strategies where inferred.

Random Forest			Euclidian Distance			Weighted Euclidian		
1000	2500	5000	1000	2500	5000	1000	2500	5000
38%	42%	41%	44%	52%	46%	55%	75%	80%
25%	50%	63%	55%	57%	67%	65%	56%	70%
50%	55%	68%	60%	66%	84%	50%	84%	86%
45%	68%	67%	70%	71%	72%	53%	69%	73%
30%	51%	56%	55%	64%	70%	47%	77%	81%
56%	77%	78%	67%	76%	77%	67%	58%	79%
50%	76%	75%	49%	51%	70%	45%	59%	78%
62%	70%	82%	30%	65%	70%	33%	81%	86%
33%	40%	50%	40%	65%	53%	50%	70%	75%
61%	64%	67%	51%	67%	71%	54%	71%	79%

7. PokerBuilder Agent

The final step of this work was to build a poker agent that uses previously created strategies. In order to be able to follow the strategies, the agent needs some reading features of the information gathered at a poker table. Obtaining evaluators' features is trivial because they comprise perfect information (data obtained just by looking at the table). Predictors, however, require a statistical study of the played hands in order to get reliable information. Another feature required by the agent is an algorithm to select which rule to apply. An agent with these features will be an agent capable of strictly following the strategy defined previously.

The agent's action sequence is depicted in Algorithm 1. The algorithm takes the current history *h* of the game, the agent player *agent* and a set of tactics *T*, and returns the amount of money to bet.

The agent starts by reading the strategy to use from the respective file. In each of the states, the agent will follow sequentially three major steps: reading all the information of the table, which includes setting the values of the evaluators, and trying to suit the imperfect information of the predictors, searching the most suitable rules for the table circumstances and choosing the rule to follow. At the end of each hand, the agent will save all the hand's information: bets from the opponents, each opponent hand (if shown), and the position of the opponent among others, to be used by the opponent modeling evaluators and predictors in future games. The agent was built to work on the LIACC Simulator described in[19]. This simulator has features that ease the construction, test and validation of the agent. Moreover, due to compatibility with the AAAI simulator, it also allows for the developed agent to participate in the annual computer poker competition without any code changes [20].

Algorithm 1 $Play(h \in H, agent \in N, T)$

Let $round : H \rightarrow \{preflop, flop, turn, river\}$ *be a function that returns a round for a given history*
Let $tactic\_action : T \times \{preflop, flop, turn, river\} \rightarrow A'$ *be a function that returns an abstracted action from the tactic or null if the action is not defined.*
Let $translate : A' \times H \rightarrow A$ *be a function that translates an abstracted action to an contextualized action of a history*
Let $bet\_amount : A \rightarrow \mathrm{N}^{+}$ *be a function that returns the amount of chips to bet for a given action*

```
if p(h) <> agent
  return -1
end if
for each t in T
  action = translate(tactic_action(t,round(h)),h)
  if action <> null
    return max(bet_amount(action), s(p(h),h))
  end if
end for each
return 0
```

8. Tests & Results

Poker is a game with elements of chance thus complicating player rating. The purpose of this work is not to build a poker agent to win against every opponent but to enable the user to define behaviors in a simple way.

All the tests were conducted in the Pre-Flop version of No Limit Texas Hold'em in head's up games. Two distinct agents were built:

- Agent PokerTron - This agent has a simple strategy (with only one tactic and five rules) but yet capable of trapping and bluffing opponents along the game. The behavior of this agent with all hands has a good variety of moves making it very difficult to read.
- Agent Hansen - This agent has a much more complex strategy than PokerTron. It contains three different tactics, used in specific circumstances, being the choice of what tactic to use based on the current stack. With a large stack, the agent will play a very loose game, practically never folding any hand pre-flop and trying to get their opponents out of the game with large bets. With a normal stack it will play more specific hands (group A and B, see Table 2), avoiding making bluffs. With a very small stack, the agent will wait for a hand A or B and goes all-in.

8.1. Behavior tests

In Table 4 we can see the percentage of rule activation for each agent, during the 10 games played. This represents the number of times each agent makes a decision based on its strategy. The fact that a strategy is defined does not imply that it will be followed every single hand. This happens because the strategy does not cover all possible circumstances that can occur in a poker game. In Table 4, we can see that agent Hansen has a higher percentage of rule activation. This means that the full area of possible circumstances is more covered in agent Hansen than it is in agent PokerTron.

Table 4. Rule Activation of Hansen and PokerTron agent

	Hansen	PokerTron
Rule Activation	64%	48%

Another important statistic is the tactic activation (Table 5). In the case of PokerTron, there is only one tactic defined, but in Hansen there are three. The "aggressive" tactic has a higher percentage (the agent won most of the simulated games), which means it had a high stack most of the times. The low stack tactic was less used because this tactic is only activated for low stacks and for hands of group A and B, which did not happen often since Hansen was almost always leading the tournament.

Table 5. Tactic Activation of Hansen Agent

	HighStack	NormalStack	LowStack
Tactic activation	56%	39%	5%

8.2. Performance tests

Hansen and PokerTron were put up against the two observing agents created by Dinis Ferreira [13] in a tournament (limited resources). Fig. 4 shows that the *PokerLang* agents ended up competing against themselves with a final victory for Hansen (the agent with a more complex strategy).

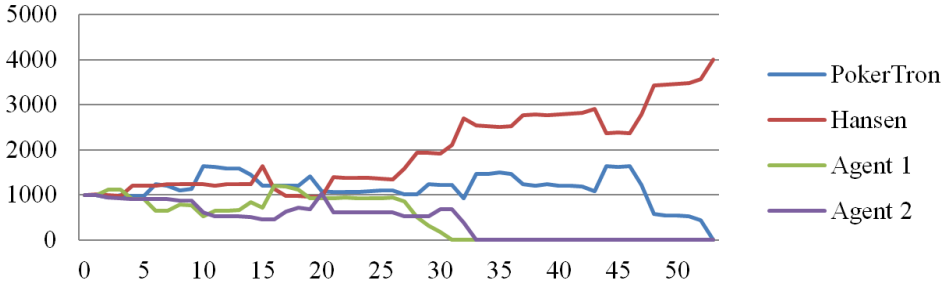

Fig. 4. Stack Evolution of the simulated games. Horizontal axis shows the number of hands and the vertical axis displays the agent's stack.

Both *PokerBuilder* agents gained advantage early in the game, being able to eliminate Agent 1 and Agent 2 in the 31st hand and 33rd hand, respectively. The most important fact to retrieve from these results is that *PokerBuilder* can be used to produce effective agents in a short time and in a very simple way. These simulations could have been made with much more games, but the purpose of these tests was to prove the efficiency of the application and the agent that supports it. The first test showed the effectiveness of the agent reading and running the strategies defined. In the Tournament simulation, the purpose is to show how *PokerBuilder* agents would behave against different agents.

9. Conclusions

The purpose of this work was to create Poker playing agents more accessible to the common user and, thus, a comprehensible high level language that represents Poker strategies was created. *PokerLang* filled the gaps of previous approaches like the Poker Programming Language because it allows for the definition of much more complex and complete strategies. An intuitive and pleasant graphical application to support the creation of *PokerLang* files was also created, thus making it easier to create playing agents. Moreover, the developed strategy inferring system proved empirically to be accurate for generating strategies similar to human ones from past played games.

Tests and simulations showed that the created agents correctly followed several *PokerLang* strategies. Moreover, agents made by Poker players were able to beat previously developed agents. However, matched between *PokerLang* agents against professional players are still required to further validate this approach.

In future research, more game concepts can be added to cover up more poker specifications and to make the agents even more effective, such as the customization of

abstraction techniques. Another important feature would be the inclusion of an exploration map to allow for the agent to assume how to play with information sets that were not defined, instead of just calling or folding. This work will also be concerned with gathering professional poker player models using this language and comparing the models with the real players' behavior in order to fully and further test the expressiveness of the *PokerLang* language.

Acknowledgments. This work was financially supported by FCT – Fundação para a Ciência e a Tecnologia through Ph.D. Scholarship SFRH/BD/71598/2010.

10. Appendix: PokerBuilder software GUI screenshots

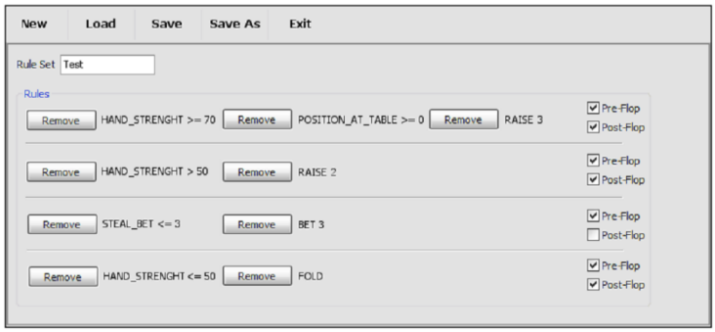

Fig. 5. Strategy builder

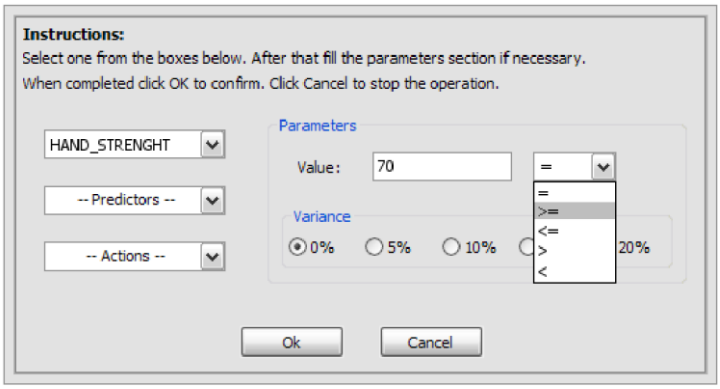

Fig. 6. Rule creator

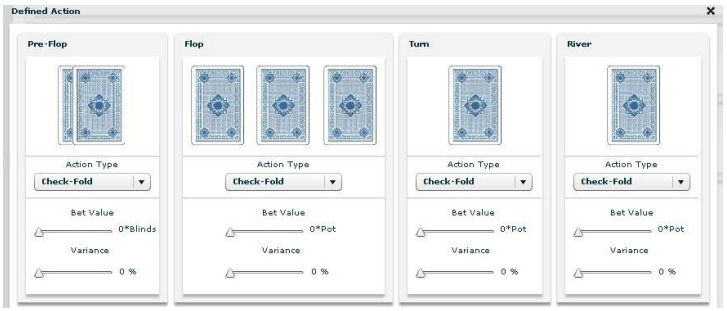

Fig. 7. Action behavior editor in *PokerBuilder*

References

1. Johanson, M.: Measuring the Size of Large No-Limit Poker Games. University of Alberta, Technical report (2013).
2. Billings, D., Papp, D., Schaeffer, J., Szafron, D.: Opponent modeling in poker. AAAI Conference on Artificial Intelligence. pp. 493–499 (1998).
3. Billings, D., Papp, D., Peña, L., Schaeffer, J., Szafron, D.: Using Selective-Sampling Simulations in Poker. AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information,. pp. 13–18 (1999).
4. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret Minimization in Games with Incomplete Information. Advances in Neural Information Processing Systems 20 (NIPS). pp. 1729–1736 (2008).
5. Johanson, M., Zinkevich, M., Bowling, M.: Computing Robust Counter-Strategies. NIPS. pp. 1128–1135 (2007).
6. Johanson, M., Bowling, M.: Data biased robust counter strategies. Proceedings ofthe Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS). pp. 264–271 (2009).
7. The Second Man-Machine Poker Competition, http://webdocs.cs.ualberta.ca/~games/poker/man-machine/.
8. Teófilo, L.F., Reis, L.P.: Building a No Limit Texas Hold'em Poker Playing Agent based on Game Logs using Supervised Learning. Proceedings 2nd International Conference on Autonomous and Intelligent Systems. pp. 73–83 (2011).
9. Rubin, J., Watson, I.: Case-based strategies in computer poker. AI Commun. 25, 19–48 (2012).
10. Broeck, G., Driessens, K., Ramon, J.: Monte-Carlo Tree Search in Poker Using Expected Reward Distributions. ACML '09 Proceedings of the 1st Asian Conference on Machine Learning: Advances in Machine Learning. pp. 367–381 (2009).
11. Teófilo, L.F., Passos, N., Reis, L.P., Lopes Cardoso, H.: Adapting Strategies to Opponent Models in Incomplete Information Games: A Reinforcement Learning Approach for Poker. Autonomous and Intelligent Systems - Third International Conference (AIS2012). pp. 220–227 (2012).
12. Rubin, J., Watson, I.: Computer poker: A review. Artif. Intell. 175, 958–987 (2011).
13. Teófilo, L.F., Reis, L.P., Lopes Cardoso, H.: Computer Poker Research at LIACC. Computer Poker Symposium. AAAI (2012).
14. Shanky Technologies: Poker Programming Language User Guide. (2009).

15. Reis, L.P., Lau, N., Springer: COACH UNILANG - A Standard Language for Coaching a (Robo)Soccer Team. In: Birk, A., Coradeschi, S., and Tadokoro, S. (eds.) Robocup 2001 Robot Soccer World Cup. pp. 183–192. Springer-Verlag (2002).
16. Harrington, D., Robertie, B.: Harrington on Hold 'em Expert Strategy for No Limit Tournaments, Vol. 1: Strategic Play. Two Plus Two Pub. (2004).
17. Teofilo, L.F., Reis, L.P., Cardoso, H.L.: Estimating the Odds for Texas Hold'em Poker Agents. 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). pp. 369–374. IEEE (2013).
18. Sklansky, D.: The Theory of Poker: A Professional Poker Player Teaches You How to Think Like One. Two Plus Two (2007).
19. Teófilo, L.F., Rossetti, R., Reis, L.P., Lopes Cardoso, H.: Simulation and Performance Assessment of Poker Agents. Springer LNCS 7838 (MABS 2012). pp. 69–84. Springer-Verlag, Valência, Spain (2013).
20. Zinkevich, M., Littman, M.L.: The 2006 AAAI Computer Poker Competition. J. Int. Comput. Games Assoc. 166–167 (2006).

Luís Filipe Teófilo holds a Master in Informatics and Computing Engineering from University of Porto since 2010. Luís is a researcher and enthusiast on artificial intelligence applied to games with several scientific publications, especially on the Texas Hold'em Poker domain. Currently he is also an Assistant Professor at University of Porto and Ph.D. researcher at LIACC.

Luis Paulo Reis is an Associate Professor at the Information Systems Department at the School of Engineering, University of Minho, Portugal and a member of the Directive Board of the Artificial Intelligence and Computer Science Lab, Portugal. He received his Electrical Engineering and MSc degrees from the University of Porto in 1993 and 1995, and a PhD in Electrical Engineering (Artificial Intelligence/Robotics) at the same University in 2003. His research interests include also Multi-Agent Systems (MAS) and Intelligent Simulation. He was principal investigator of more than 10 research projects in those areas and he was evaluator for the European Commission for FP6 Projects. He is the team leader of FC Portugal robotic soccer team/project, three times World Champion and seven times European Champion in RoboCup. He also won more than 30 scientific awards, including best papers at several conferences such as ICEIS and Robotica.

Henrique Lopes Cardoso is an Assistant Professor of the Department of Informatics Engineering at the Faculty of Engineering of the University of Porto (FEUP), and a researcher at the Artificial Intelligence and Computer Science Lab (LIACC). He received a M.Sc. degree on Artificial Intelligence in 1999 and a PhD on Informatics Engineering in 2011, both at FEUP. His main research interests lie within multi-agent systems research, focusing on social aspects such as agreements, norms and trust, but also on simulation and development tools.

Received: September 21, 2013; Accepted: April 01, 2014