A Profitable Online No-Limit Poker Playing Agent

Abstract— **The No-Limit Texas Hold'em variant of Poker is the game that is most frequently used to assess new developments in incomplete information problems, through the development of game playing agents. For this particular game, current state-of-the-art techniques consist in the pre-computation of a set of strategies that are in a Nash-Equilibrium state. However, due to the game's decision tree size, current algorithms only work in an abstracted version of No-Limit Poker. Moreover, since these strategies are static, they ignore the opponents' playing style thus being unable to maximize profit against certain kinds of opponents. This makes these strategies unusable when playing in an online environment against human players. In this paper we present a rule-based strategy approach for a No-Limit Poker agent that was developed to play online, against human players and in online multiplayer matches. This strategy is based on a popular technique used by human players – short stack playing – which consists of playing in tables with up to 6 players and low initial resources. Using domain specific opponent modeling techniques and limiting the decisions to the first round of the game, the agent was able to make a good profit margin of 11.5% per game when playing against human players. The significance of our results resides in the fact that, for the first time in the Computer Poker literature, we present a game playing agent that can match human players in multiplayer games.**

Keywords—poker; rule-based strategies; abstraction; opponent modeling; game playing agents

I. INTRODUCTION

Games of incomplete information such as Poker are a popular domain of research in the area of artificial intelligence. Poker presents unique challenging problems, such as opponent modeling, risk management and bluff detection. The development of software agents that can compute probabilistic decisions considering those problems is a difficult task, since dynamic and live adaption to the opponents' strategies is required in order to create a robust computer Poker agent.

Several important developments were accomplished in the domain of Computer Poker in the last years. Most popular techniques are based on game-theory concepts: the agent uses a pre-computed and static set of strategies that approximate a Nash-Equilibrium [1][2][3]. While these strategies are robust against a unknown opponent, they present two problems: they do not try to maximize profit against a certain player and current algorithms are only proved to converge to a Nash-Equilibrium in one-on-one games. These limitations are very important in multiplayer games because the most profitable player can be, for instance, the one who earns more money from the worst player. This means that in Poker it is not enough to beat a certain player; the goal of the game is to actually maximize the average earnings per game against all players. Another example: if one can beat a very good player that does not necessarily make him or her good and profitable.

Despite the good results achieved by Nash-Equilibrium agents, where one managed to defeat a very good human player in a one-on-one tournament [4], until now there is no hard evidence on how these agents would perform in multiplayer games with human players. This aspect is essential, since real Poker games can hold up to 10 players in a table, and playing against several opponents is far more challenging than playing against a single opponent.

In this paper we present, for the first time, a Poker agent strategy approach that shows empirical evidence that it can beat human players in online multiplayer tables. Our agent's approach is a rule-based strategy that originated from the domain experts' short-stack strategy concept. Opponent models are computed online and are based on opponents' features such as their aggressiveness, position at the table and willingness to bet. These features are combined with the agent's perspective view features such as the number of players involved and the mathematical expectation of the agent's score (the set of cards that it holds) to define the information set and therefore which action is mapped to it.

The paper's main contribution resides on the fact that, to the best of our knowledge, we present the first reported matches that oppose software agents and human players, in real money games, in multiplayer tables. Our results can be considered significant because the agent was relatively victorious – the agent was able to be marginally profitable in low stake[1] games (0.02/0.01 cents). In order to avoid playstyle biases, the experiments were performed without human players' awareness that they were competing against a software agent.

The rest of the paper is organized as follows. Section II presents this work's background and some notation used throughout the paper. It also summarizes the game of Poker by briefly presenting its rules and key concepts that are necessary

[1] Stake – minimum bet value.

to understand the agent's implementation. Section III presents related methodologies and recent developments in the Computer domain. Section IV describes the implemented agent's strategy approach and its integration with the required supporting tools. Section V presents and discusses agent's performance when playing online against human players. Finally, Section VI concludes this paper and points out directions for future research and possible enhancements that can be implemented in the agent's strategy.

II. Poker Background & Notation

In this section we provide information about the game of Poker, with emphasis on the variant that was used for the experiments in section V – No-Limit Texas Hold'em Poker – which is considered to be the most popular for professional players and one of the most challenging for computer scientists.

Throughout the paper Poker will be represented as an extensive-form game. An extensive-form game is 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 denoted by h considering that $h \in H$, being H the set of all possible game sequences according to the game's rules. We denote h' as a particular history-prefix where $h \subseteq \mathrm{h}'$. Therefore, a game of Poker G can be represented as the following tuple:

$$G = \langle H, Z, N, \mathrm{A}, a\colon H \rightarrow \mathrm{A}, \mathrm{p}\colon \mathrm{H} \rightarrow \mathrm{N}, u\colon N \times Z \rightarrow \mathbb{R}\rangle | Z \subset H$$

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 all players in the game G and A is the set of all possible actions.

An extensive-form game also requires the definition of three functions. The function a gives the set of all possible actions for a given node (or history) where for any particular node $z \in Z \Rightarrow a(z) = \emptyset$ and $h \in H \backslash Z \Rightarrow a(h) \neq \emptyset$. 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.

A. *Scoring*

At the beginning of a game G, each participant 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 $\forall i \in N \wedge j \in N\colon P_i \cap P_j = \emptyset$. The private cards P_i are only visible to player i and may never be unveiled to other players (only if the game reaches a shodown[2]). At certain moments of the game, some shared cards are revealed – we will denote S the set of all shared cards and S_r the set of visible shared cards at round r where $\forall i \in N, r\colon S_r \subset S \subset D \wedge \mathrm{S}_r \cap P_i = \emptyset$. The shared cards are always visible to every player and are used in combination with the private cards to determine a particular player's score.

In Poker, the utility of a player i is given by $w \in [P_i \cup S]^5$ where $s(w)$ is maximized, being $s\colon [D]^5 \rightarrow \mathbb{N}_{\geq 0}$ a function that returns the score of a 5 card set. Therefore, for any remaining players i, j, player i wins against j if $\max_{w \in [P_i \cup S]^5} score(w) > \max_{k \in [P_j \cup S]^5} score(k)$. 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 each is divided in several sub-ranks. The total number of sub-ranks is 7462, therefore $\forall w\colon score(w) \in [0,7461]$.

B. *Rules and utility*

After the cards are dealt to each player, two of them post the blinds – minimum bet values – and then 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[3]:

- Call – match the highest bet. If the call costs 0, then it is known as Check.
- Raise – increase the highest bet. If this action costs the full player's stack, it is known as All-In.
- Fold – forfeit the game

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 ends, the player with the highest ranked set of cards wins the game and collects the pot, as explained in 2.1. 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 hand of lower score. This particular feature of the game's rules makes it difficult to assess an agent's decision. Regardless the winning situation, $\forall z \in Z\colon \sum_{i \in N} u(i,z) = 0$, making Poker a zero-sum game. However, usually in online Poker the game is not zero-sum due to the rake – 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, since the tests were performed in an online playing room, the e takes an important role in the results. In our experiments, $e \in [0.025, 0.05]$.

In order to complete the definition of a Poker game, we define it as G_P.

$$G_P = \langle H, Z, N, \mathrm{A}, \mathrm{P}, \mathrm{S}, a, \mathrm{p}, u,$$
$$s\colon N \times H \rightarrow \mathbb{Q}_{\geq 0}, b\colon N \times H \rightarrow \mathbb{Q}, r\colon H \rightarrow R, c\colon H \rightarrow \mathbb{Q}_{\geq 0},$$
$$v\colon H \rightarrow S_r\}\rangle | Z \subset H \wedge R = \{n\colon n \subset N\} \wedge S_r \subseteq S$$

First, the sets P and S (see II.A) were included and they respectively correspond to the private and community cards sets ($\forall i\colon P_i \in P$). The functions s, b, c, v and r were added to the original definition of G. Function s denotes the amount of remaining cash (stack) and b the amount of cash betted by a

[2] Showdown – a game's terminal node with at least 2 standing players and all bets matched.

[3] Pot – accumulated value of all bets.

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 $\sum_i^N b(i,h)$ then, given Texas Hold'em rules, the player's utility in a terminal node z:

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

Given this definitions we can also detail the a function. The No-Limit variant of Texas Hold'em Poker is characterized by having no limited betting – the players can raise up to their total stack value:

$$\forall h \in H: a(h) \in \begin{bmatrix} \min\big(s(p(h),h), c(h) - b(p(h),h)\big), \\ s(p(h),h) \end{bmatrix} \cup \{0\} \wedge A = \mathbb{Q}_{\geq 0}$$

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 is all-in.

III. Related Work

The first successful approaches to create Poker agents were rule-Based strategy definitions, which involves specifying the action that should be taken for a given information set [5]. 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 $I_{i,h} = \{h, P_i, v(h)\} | I_{i,h} \in I$ are composed by the game's action sequence, the player's private cards and the visible community cards. Other features can be extrapolated from h. *Pokerlang* [6] is a useful domain language that simplifies the specification of such strategies. Despite the simplicity of this technique, it is still successful for more complex Poker variants like multiplayer No-Limit Texas Hold'em [7].

The next approaches were based on simulation techniques like [8], i.e. generating game random 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) in [1]. The CFR algorithm allows for the computation of a Nash Equilibrium approximation strategy in large games such as Poker through self-play, for two players. This could be done before through linear programming methods (e.g. 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). Several approaches based on CFR, like Restricted Nash Response [9] and Data-biased response [10] backed up the first victories against Poker experts [11]. The main issue about CFR is that it only proved to converge to a Nash-Equilibrium for two players' games. The strategies generated for more than two players are, however, robust but the obtained results present a large variance – in some cases the CFR strategy performs better against good players and worse against bad players. Another problem is that these types of strategies are static which means that they are unable to dynamically adapt to changing game conditions.

Other recent methodologies based on pattern matching [12] and cased based reasoning [13]. 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, the described approaches based their strategies on the concept of information set similarity. In [13], two information sets have a degree of similarity equal to the average similarity of the game features. In [12], instead of the average, the degree of similarity was measured through the Euclidean distance between the game features. The Monte Carlo Search Tree algorithm [14] and reinforcement learning approaches [15] are other techniques that were successfully applied to the domain of Computer Poker. One should not also forget some work done in opponent modeling techniques, namely [16]. A more throughout description of the most recent works can be found in the reviews [17], [18].

IV. Agent Implementation

In this section we demonstrate the methodology that was followed to implement the game playing agent. Our development approach was divided in three phases:

Online room interface – an interface which allows for Poker playing agents to impersonate a human player. In other words, this interface recognizes what is going on in a Poker room, provides de information to the software agents, receives the agent's response and finally controls the mouse and the keyboard to play accordingly to the agent's desire.

Extracting opponent models – this consists in observing the opponents actions and label each one with a strategy type. The action of our agent's strategy depends on the types of strategies of the current opponents. An external tool called Hold'em Manager[4] is used for this phase.

The agent's strategy, which is based on a rule-based strategy from an expert player. This module is completely independent of the aforementioned, i.e. the agent can provide outputs and receive inputs from different platforms. This allows for testing the agent in a simulation environment, against other previously developed agents, without any extra effort. This was important to reduce the costs of our tests because, as mentioned earlier, the experiments described in this paper were performed online in real money games.

The image on fig. 1 summarizes the global view of our agent and how the different components communicate. The decision workflow is an endless cycle, i.e. the agent keeps reading events from the table. The cycle is interrupted when the agent is unable to read from the Poker Game UI which causes a timeout in the *"Read an event from the game UI"*.

[4] Hold'em manager website: http://www.holdemmanager.com/

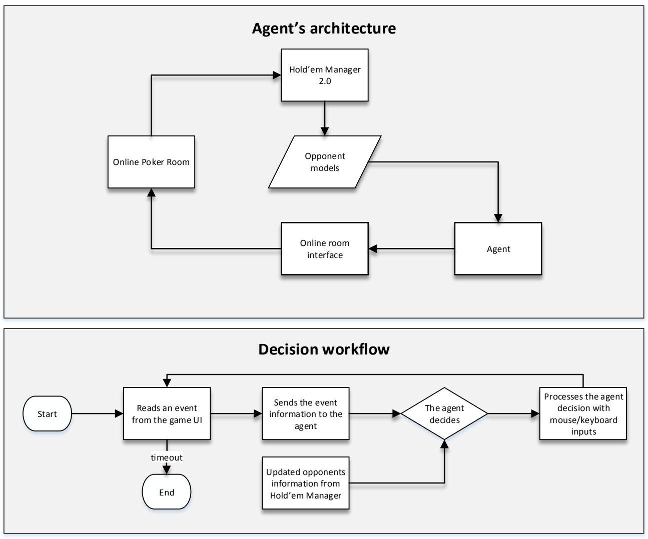
Agent's architecture
Hold'em Manager 2.0
Online Poker Room
Opponent models
Online room interface
Agent
Decision workflow
Start
Reads an event from the game UI
Sends the event information to the agent
The agent decides
Processes the agent decision with mouse/keyboard inputs
timeout
End
Updated opponents information from Hold'em Manager

Fig. 1. The agent's architecture and decision workflow.

A. *Extracting opponent models*

The opponent models are based on three common statistics about the players (VPIP, Fold3Bet and PFR). These statistics are collected during the games. The more we play against a certain player, the more these statistics will reflect his or her playing style.

- $vpip(opponent \in N)$ – this statistic value stands for "Voluntarily Put $ In Pot" and tells the percentage of times a player makes a call or a raise on pre-flop round.
- $fold3bet(opponent \in N)$ – this statistic value tells the percentage of times a player raises and folds to a re-raise. It is possible to know the fold to 3bet for any position at the table. That value will be useful in order to calculate if the expected return is positive or negative against the hand the agent holds.
- $pfr(opponent \in N)$ – this statistic value tells the percentage of times a player raises a hand pre-flop

All these statistics are computed by the Hold'em Manager software and store in a relational database. The agent extracts these by a direct connection to the database.

B. *The agent's strategy*

Let's consider $hero \in N$ being the developed agent playing a particular G_P. The developed agent follows a short-stack strategy. A short stack strategy has the following characteristics:

- Playing with a money stack (money brought to the game) of at most 20 big-blinds (minimum bet value).$\forall h: s(hero, h) + b(hero, h) \leq 20 \times c(h_0)$, being $h0$ the history of the first game decision.
- Initial number of opponents between 4 and 6. $5 \geq |N| \geq 7$.
- Decisions are limited to the Pre-Flop round, knowing that $|S_{flop}| = \emptyset$, which means that the decisions only consider the hero's private cards.
- Hero's decision abstraction. Hero only chooses from three possible actions – fold, call and all-in – ignoring therefor all possible raise values. The call action is only used if the hero decides to fold when the call action is free. In short, the possible decisions are $\forall h \in H \wedge p(h) = hero: a(h) \in \{0, s(p(h), h)\}$.

Before describing the algorithm, it is important to describe how to compute the equity (algorithm 1).

Algorithm 1 $Equity(h \in H, hero \in N, perc, niter)$

```
$win = 0$
$tie = 0$
$lose = 0$
$iter = 0$
$Let\ pairs = the\ list\ of\ the\ possible\ card\ pairs,\ ordered\ by\ value$
$pp = sublist(pairs, (1 - perc) \times len(pairs), len(pairs))$
for each $p$ in $pp \backslash P_{hero}$
    while $iter < niter$ do
        $Let\ board = gen\_random\_board(D \backslash p \backslash P_{hero}, 5)$
        $ourrank = \max_{w \in [P_{hero} \cup board]^5} score(w)$
        $opprank = \max_{w \in [p \cup board]^5} score(w)$
        if $ourrank > opprank$ then $win{+}{+}$
        else if $ourrank < opprank$ then $lose{+}{+}$
        else $tied{+}{+}$
        end if
        $iter{+}{+}$
    end while
end for each
return $\left(1 - \frac{lose}{win+tie+lose}\right)$
```

The equity is the probability of a certain player's hand winning when dealing the remaining hidden shared cards. Since we are making Pre-Flop decisions, where no shared were yet revealed, we have to randomly generate possible shared cards. The same happens for opponents' private cards, because they remain hidden the whole game (and they might not even be revealed at all). The algorithm for the equity computation is as follows. It uses a Monte Carlo sampling approach to reduce the computation time i.e. instead of generating all possible shared card samples, it uses a fixed number of possible boards. As for the opponent card sampling, we consider the variable $Perc$ as input. $Perc$ indicates the percentile of the strength of possible opponents' starting hands. For instance, if $Perc = 28\%$, it means that we consider that our opponent is only likely to have the best 28% starting hands.

The next step is to evaluate the game state. The game state evaluation considers the number of players that have called $\#callers$, the number of players that have raised $\#raisers$ and the number of players that are all-in $\#alliners$. Table I indicates the possible abstracted game states.

TABLE I. POSSIBLE GAME STATE ABSTRACTIONS CONSIDERED BY HERO.

State	$\#callers$	$\#raisers$	$\#alliners$
unopened	0	0	0
limped	1	0	0
raised	0	1	0
allin	0	0	1
limps	>1	0	0

Next, we need to classify the hero's starting hand strength. For this, we need two measures: the hand classification function $hclass: D^2 \rightarrow \{1,2,3,4,5,6,7,8\}$, given by Table II and the expected hand return given by algorithm 2.

```
Algorithm 2 $ExpectedReturn(hero \in N, opponent \in N, h \in H)$
    $Let\ f3b = fold3bet(opponent)$
    $Let\ bbs = vpip(opponent)$
    $Let\ eq = Equity(h, hero, bbs, 10000)$
    $Let\ h_0$ be the prefix of $h$ where $|h_0| = 0$
    $Let\ pot = \sum_{i}^{N} b(i, h)$
    return  $\Big(\big((f3b - |N| \times c(h_0) \times pot) + \big((1 - f3b) \times (eq) \times (bbs + pot)\big)\big) - \big((1 - eq) \times (bbs + pot)\big)\Big)$
```

TABLE II. STARTING CARDS CLASSIFICATION. 1 FOR TOP SCORED HANDS AND 8 FOR LOW SCORED HANDS. HANDS WITHOUT CLASSIFICATION IN THIS TABLE ARE CONSIDERED UNPLAYABLE THUS THE HERO FOLDS IMMEDIATELY WHEN IT HOLDS SUCH HANDS.

		Offsuit												
		A	**K**	**Q**	**J**	**T**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Suited	**A**	1	1	2	2	3	5	5	5	5	5	5	5	5
	K	2	1	2	3	4	6	7	7	7	7	7	7	7
	Q	3	4	1	3	4	5	7	■	■	■	■	■	■
	J	4	5	5	1	3	4	6	8	■	■	■	■	■
	T	6	6	6	5	2	4	5	7	■	■	■	■	■
	9	8	8	8	7	7	3	4	5	8	■	■	■	■
	8	■	■	■	8	8	7	4	5	6	8	■	■	■
	7	■	■	■	■	■	■	8	5	5	6	8	■	■
	6	■	■	■	■	■	■	■	8	6	7	7	■	■
	5	■	■	■	■	■	■	■	■	8	6	6	7	■
	4	■	■	■	■	■	■	■	■	■	8	7	7	8
	3	■	■	■	■	■	■	■	■	■	■	■	7	8
	2	■	■	■	■	■	■	■	■	■	■	■	■	7

Finally, we present the game playing algorithm – algorithm 3. This algorithm uses a rule-based approach that considers the abstracted game state, and the expected return of the current hand, in order to decide either to fold or go all-in. It returns the bet value.

V. RESULTS

Given that our agent implementation only plays in a single table at a time and given that the agent was playing against humans, the result extraction is very time consuming. Even so, we were able to extract the results of 3814 games in online games (see some statistics in Table III).

The overall profit of the agent was 1.48 big-blinds (minimum bets) for each 100 games. Since we performed the experiments in tables where the blinds were 0.02€, the agent made an overall absolute profit of 1.13€. Considering that in each game the agent had to pay an average 5% commission over the betted money, these results can be considered good. Moreover, this particular online casino refunds 20% of the money paid on commissions, at the end of the month, when the player is profitable. This allowed for the agent to make an extra absolute profit value of 7.63€, making a total profit of 8.76€. This results in a final average profit of about 11.5 big-blinds for each 100 games.

```
Algorithm 3 $Strategy(hero \in N, h \in H)$
    $Let\ foldOrCall = 0$
    $Let\ allin = s(p(h), h)$
    $Let\ opp =$ the last playing opponent that went all-in. If none, select the last playing opponent that raised. If none, select the last playing opponent. If none, select the player in the dealer position.
    $Let\ pos =$ the hero's position in table. It can be bb (if the hero is the big-blind), sb (the small-blind position), btn (hero is the dealer – last to act), co (cut-off position – before dealer) and utg (under the gun position – first to act).
    $Let\ pos_{opp} =$ the opp position in table (with the same possible values as the hero's position).
    $Let\ er = ExpectedReturn(hero, opp, h)$
    $Let\ gameState =$ the game's state according to Table I.

    if $hclass(P_{hero}) = 1$ then
        return allin
    else if $hclass(P_{hero}) = 2 \wedge er \geq 0$ then
        switch gameState
            case unopened
                if $pos = sb$ then
                    return rand_real_between(0.0, 1.0)>0.4?allin:fold
                else if $pos = co \vee pos = btn$ then
                    return allin
                end if
            case limped ∨ allin
                if $pos = bb \vee pos = sb$ then
                    return allin
                end if
            case limps
                if $pos = bb$ then
                    return allin
                end if
            case raised
                if $pos = bb \vee pos = sb \vee pos = btn$ then
                    return allin
                end if
        return foldOrCall
    else if $hclass(P_{hero}) = 3 \wedge er \geq 0$ then
        switch gameState
            case unopened
                if $pos = btn \vee pos = sb$ then
                    return allin
                end if
            case limped ∨ raised
                if $pos = bb$ then
                    return allin
                end if
        return foldOrCall
    end if

    if $pos = bb \wedge gameState = raised \wedge$
                $(pos_{opp} = btn \vee pos_{opp} = sb) \wedge$
                    $fold3bet(opp) \geq 0.5$ then
        return allin
    else if $pos = bb \wedge gameState = raised \wedge$
            $pos_{opp} = btn \wedge fold3bet(opp) \geq 0.5$ then
        return allin
    end if

    return foldOrCall
```

A. *All-time results*

A graphical representation of the hands played and the agent's profit balance overtime is shown on fig. 2. In this chart we consider that the commission refund function is linear.

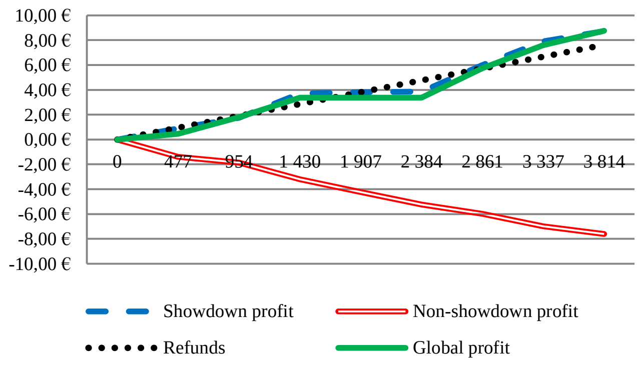

Fig. 2. Agent's all time profit.

As can be observed on fig. 2, the agent's total money balance increases overtime, ending up in a final absolute profit of 8.76€. In this graph, besides de the global profit and the commission refunding profit, we also indicate the showdown and non-showdown profit. The showdown profit includes money lost or won in all games where the agent decided to bet and at least one of the opponents covered that bet. Non-showdown profit includes all money lost when the agent folds or all the money won when the agent goes all-in and all opponents fold.

A conclusion that can be taken from this graph is the importance of stealing and defending blinds (see sub-section D). Since the agent is a tight player (it only plays a small number of hands), it ends up folding 0.02€ or 0.01€ too many times, when it is the blinds position. This results in losing too much money (Non-showdown winnings). The only way to reduce these loses would be to play in other rounds instead of Pre-Flop. Being a less tight agent would probably reduce the showdown games earnings.

However, it is possible to observe a slight difference on the non-showdown line, after the 2800 hands, where the gradient starts to decrease. The reason behind this is the gradual improvement of the agent's evaluation on the opponents' pre-flop steal ability. We believe that the results will improve when even more games are played. However, the profit already made by the agent in the showdown winnings compensates its lack of defending blinds ability.

B. *Playing style*

In order to analyze the agent's playing style, we demonstrate on Table III some relevant statistics that summarize the agent's online performance on this experiment. These statistics do not include the commission refunds. Now, we describe each statistic:

- Number of games – the total number of Poker games played in this experiment.
- VPIP (Voluntary Put money In Pot) – indicates the percentage of games where the agent bets thus excluding the money betted when the agent was in the blinds positions. As expected and as said earlier, since the agent's strategy is tight, the agent only went all-in in 9% of the games.
- PFR (Pre-flop raise) – number of times the agent raises any amount in the Pre-Flop round. Since the agent's strategy only considers the Pre-Flop round, this value is very similar to VPIP. The agent only plays after the Pre-Flop if it can get a free Flop, which means that the agent is in the big-blind position and none of the opponents bet any amount thus enabling the agent to just call the hand.
- 3Bet – the number of times the agent raises after any opponent has raised. As expected, this measure is also similar to VPIP since the agent usually only plays in table positions where it decides the action after other players.
- Winnings – the absolute winnings excluding the commission refunds. These winnings, depending on the value of the blinds, are the ones that indicate if the agent is entitled to commission refunding.
- Bb/100 games – the number of big-blinds (minimum bets) won for each 100 games. This is the common measure that is used to evaluate if a player is good or not. The way the value of this measure has to be looked depends greatly on the value of the blinds. For instance, for games with blinds of 0.50-1.00€, a good player should have about 7Bb/game. In 0.02-0.01€ games, a good player should have about 10Bb/game.
- Avg. All-in EV – the expected value when the agent's goes all-in. This measure is relative to the investment made by the agent. In these experiments, the average EV is 54.6%, which means that when the agent goes all-in, it has a positive profit of 54.6% of the amount that was betted. For this stat, we indicate its average value in the game, in the Pre-Flop and after the Pre-Flop.

TABLE III. AGENT'S PLAYING STYLE STATISTICS.

Feature	Value
Number of hands	3814
VPIP	9.3
PFR	9.0
3Bet	8.9
Winnings	1.13€
Bb/100 games	1.48
Avg. All-in EV	54.6%
Avg. Pre-flop All-in EV	54.3%
Avg. Flop All-in EV	57.0%

From these stats we must highlight the positive expected value for all-in actions in all rounds. This means that, when the agent's goes all-in, it profits in average more than 50% of its investment.

C. *Playing with table position*

Now we analyze the agent's ability to play in different position in the table (Table IV).Again we are not considering the profit made from refunds. The meaning of the position on table IV is:

- Small-blind – the player has to pay 0.01€ at the start of the game without seeing its cards. It is the penultimate player to choose his/her action.
- Big-blind – the player has to pay 0.02€ at the start of the game without seeing its cards. It is the last player to act.
- Early – no blinds. It is one of the first players to act. This position is disadvantageous because the player has to act without any feedback from his/her opponents.
- Button – also known as dealer position. In the Pre-Flop is antepenultimate player to act or the last if only two players are playing. It is the most advantageous position since the player does not have mandatory bets and he/she can get feedback from the actions of most of the opponents.
- Cutoff – position just before the button.
- Middle – positions between the last early and the cutoff.

TABLE IV. AGENT'S PLAYING STYLE STATISTICS.

Position	Hands	Profit	EV	VPIP%	PFR%	3Bet%
Small blind	695	-1.43€	-2.80€	14.0%	13.5%	11.5%
Big blind	701	-4.47€	-4.73€	10.4%	10.1%	9.5%
Early	411	1.54€	1.86€	5.6%	5.6%	-
Middle	620	0.77€	1.34€	6.8%	6.8%	6.5%
Cutoff	685	-0.34€	1.18€	7.2%	6.9%	5.5%
Button	702	5.06€	3.17€	10.0%	9.5%	6.9%
Totals	3814	1.13€	0.02€	9.3%	9.0%	8.9%

The conclusions that we can take from these results are that playing in positions where blind bets are made, will always pose the threat of losing money; the only way to lessen this leak is to improve the evaluation on the stealing probability. The agent's performance in each position is overall satisfying, showing a profit on almost all positions excluding the blinds and the cutoff. The cutoff negative income is probably due to the results' variance (low number of games), since the expected value in that position is positive. A very satisfying statistic to highlight is the average all-in percentage which is above 50% in all positions. This surely proves that the more hands the agent plays the more profit it will attain. Taking in mind the small blind VPIP being the highest among all, means that the agent tries to steal the big blind every chance he sees fit. Also the highest average all-in percentage comes from the early position, which is expected since it is the position the agent plays more seldom, making its hand ranges a lot stronger. We can see a direct connection between playing more hands, higher VPIP; and having less chances of winning, average all-in percent, but also higher chances of winning by stealing blinds.

D. Stealing and defending blinds

In table V we demonstrate the agent's results when defending and stealing blinds situations (without profit refunds).

Stealing blinds is a situation where the agent raises at Cutoff, Button or small-blind positions. The stealing blind results are extremely positive, since the agent's objective is to steal blinds while taking into account the fold chance of the opponents, since it does not play in other rounds. When the steal attempt fails, the most common scenario is probably the fact of the agent being behind the opponents range, but this is not verified, since the average all-in percentage when the agent fails to steal is higher than 50%, thus meaning the agent is still stealing less than he should be. The possibility for opening the range of stealing and having an average all-in percentage between 48% and 49% is still viable since some money is won when the opponents fold. These results show the high importance the steal factor has in the poker game (3.59€ in only 2.67% of the games has a huge significance). On Fig. 3 we can observe the positive growing rate (about 3.5%) of the profit in these situations.

TABLE V. DEFENDING AND STEALING BLINDS RESULTS.

Type	Hands	Profit	EV	Avg All-in EV%
Stealing	102	3.59€	1.81€	55.1%
Defending	51	0.51€	1.60€	49.3%

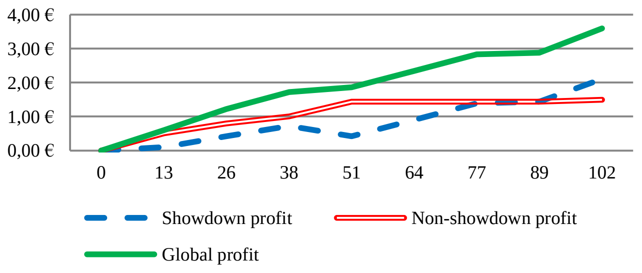

Fig. 3. Stealing blinds results.

Defending a blind is a situation where the agent is in a table position where it has to bet blinds, and has to reply to a raise from another player. On Table V we demonstrate the results when the agent tries to defend the blind by going all-in. Here we see that the calculations for expected value where fairly accurate since among all the all-ins made, the average all-in expected value percentage is 49.3%, meaning when the agent is called it will still have a good winning rate, and when it doesn't get called it wins the blinds plus the raises of the opponents. The expected value from these plays is higher than the actual winnings, this mean that the agent played well, despite of the variance not being on his side. Nevertheless it is still a small amount of hands, and in the long run the winnings will even out with the expected value. It is a very satisfying expected value of 1.60€, since the agent bets 0.20€ at a time.

We can conclude by these results that when the agent defends the blind, it defends it correctly. However, by looking again at the results on Fig. 2, we can assert that the agent either just doesn't defend the blinds enough times or that no more profit can be made from this situation.

E. Results against particular players

In this section we present the agent's results against several particular players.

On Table VI, we demonstrate the results against the players that allowed the agent to make more profit. The most significant players to note here are the ones which have a number of hands higher than 100, namely: Player2, Player9 and Player10. These three players show fairly good statistics, making them tight aggressive players (most winning players are tight aggressive), and still the agent was able to exploit them and make a positive profit over time.

TABLE VI. RESULTS AGAINST THE 10 MOST PROFITABLE OPPONENTS

Opponent	Hands	VPIP	PFR	Profit
Player1	14	42.9%	28.6	1.49€
Player2	271	17.7%	13.7%	1.00€
Player3	14	64.3%	21.4%	0.97€
Player4	41	82.9%	9.8%	0.79€
Player5	39	41.0%	12.8%	0.76€
Player6	5	40.0%	0.0%	0.73€
Player7	16	31.3%	18.8%	0.69€
Player8	45	57.8%	0.0%	0.62€
Player9	455	24.2%	11.2%	0.60€
Player10	860	19.8%	16.3%	0.56€

On Table VII we show the results of the players who gave negative profit to the agent. Looking at the top five most unprofitable opponents, their stats vary from a very tight aggressive player, Player5, to a very loose aggressive player, Player3. A quick look at the hands played against these players allowed us to verify that some of these opponents (Player5 or Player10) have dominated the agent's strategy. Others, like Player1 or Player6 are due to the results variance.

TABLE VII. RESULTS AGAINST THE 10 LESS PROFITABLE OPPONENTS

Opponent	Hands	VPIP	PFR	Profit
Player1	54	55.6%	25.9	-1.07€
Player2	180	31.1%	12.2%	-0.86€
Player3	38	84.2%	23.7%	-0.62€
Player4	148	26.4%	23.0%	-0.60€
Player5	277	27.8%	19.9%	-0.59€
Player6	67	22.4%	20.9%	-0.59€
Player7	136	20.6%	16.2%	-0.56€
Player8	224	20.5%	19.2%	-0.56€
Player9	25	72.0	40.0	-0.46€
Player10	774	15.0%	13.0%	-0.46€

VI. CONCLUSIONS

As stated before, this work required background knowledge and expertise of a domain expert on the Texas Hold'em variant of Poker. Nevertheless, despite the strategy not being (yet) as good (profitable) as the one from the original player, we believe we took a great step towards the goal of making Poker agents more profitable than the best human players, by showing that it is now possible to create a winning agent. The most surprising aspect was the agent surpassing most of the human players found online, just by considering the Pre-Flop stage of the game. Some suggestions for possible improvements would be working on the blind stealing ability on the three positions fit to do so: big blind, small blind and button. The agent can also be improved in the matter of autonomy at the tables, for instance, leaving a table when holding more than 20 big blinds, entering a new table where the minimum players is 4, leaving a table when it falls below 4 players thus optimizing the short-stack strategy (which is proved to work better in tables with 4 to 6 players). In future work the agent should also be tested in games with higher stakes, since they usually present more skilled players. Another important feature to add is the ability to play in simultaneous tables to allow for the agent to get profit much faster.

ACKNOWLEDGMENT

We would like to thank to FCT – Fundação para a Ciência e Tecnologia (the main national institution that finances academic research in Portugal) for supporting this work by providing the first author's Ph.D. scholarship with reference SFRH/BD/71598/2010.

REFERENCES

[1] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret Minimization in Games with Incomplete Information," in *Advances in Neural Information Processing Systems 20 (NIPS)*, 2008, pp. 1729–1736.

[2] 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.

[3] 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.

[4] University of Alberta, "The Second Man-Machine Poker Competition," 2008. [Online]. Available: http://webdocs.cs.ualberta.ca/~games/poker/man-machine/.

[5] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, "Opponent modeling in poker," in *AAAI Conference on Artificial Intelligence*, 1998, vol. pp, pp. 493–499.

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

[7] L. F. Teófilo, R. Rossetti, L. P. Reis, and H. Lopes Cardoso, "Simulation and Performance Assessment of Poker Agents," in *Springer LNCS 7838 (MABS 2012)*, 2013, pp. 69–84.

[8] 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.

[9] M. Johanson, M. Zinkevich, and M. Bowling, "Computing Robust Counter-Strategies.," in *NIPS*, 2007, pp. 1128–1135.

[10] M. Johanson and M. Bowling, "Data biased robust counter strategies," in *Proceedings ofthe Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009, pp. 264–271.

[11] "The Second Man-Machine Poker Competition," *University of Alberta*, 2008. [Online]. Available: http://webdocs.cs.ualberta.ca/~games/poker/man-machine/.

[12] L. F. Teófilo and L. P. Reis, "Building a No Limit Texas Hold'em Poker Playing Agent based on Game Logs using Supervised Learning," in *Proceedings 2nd International Conference on Autonomous and Intelligent Systems*, 2011, pp. 73–83.

[13] J. Rubin and I. Watson, "Case-based strategies in computer poker," *AI Commun.*, vol. 25, no. 1, pp. 19–48, 2012.

[14] G. Broeck, K. Driessens, and J. Ramon, "Monte-Carlo Tree Search in Poker Using Expected Reward Distributions," in *ACML '09 Proceedings of the 1st Asian Conference on Machine Learning: Advances in Machine Learning*, 2009, pp. 367–381.

[15] L. F. Teófilo, N. Passos, L. P. Reis, and H. Lopes Cardoso, "Adapting Strategies to Opponent Models in Incomplete Information Games: A Reinforcement Learning Approach for Poker," in *Autonomous and Intelligent Systems - Third International Conference (AIS2012)*, 2012, pp. 220–227.

[16] D. Félix and L. P. Reis, "Opponent Modelling in Texas Hold'em Poker as the Key for Success," pp. 893–894, Jun. 2008.

[17] J. Rubin and I. Watson, "Computer poker: A review," *Artif. Intell.*, vol. 175, no. 5–6, pp. 958–987, 2011.

[18] L. F. Teófilo, L. P. Reis, and H. Lopes Cardoso, "Computer Poker Research at LIACC," in *Computer Poker Symposium*, 2012