Perudish: A Game Framework and Modified Rule-set for Perudo

Daniel Livingstone
School of Computing
University of Paisley
High Street
Paisley, Scotland
daniel.livingstone@paisley.ac.uk

Abstract

Following the lead of (Macleod 2004b) I look at the simple dice game Perudo. Presenting an even simpler variant, I consider the probabilities of the game and present some simple heuristics for play. A simplified version of Perduo is implement and a small number of simple robot players are then built and played against each other and human players and the results presented and discussed. Finally, further research directions and extensions are considered.

Introduction

Perudo is a relatively simple competitive dice game of bluff and judgement. A number of players each have a hand of dice which they roll in secret. Players then have to make bids based on their estimates of the results of all dice held by all players.

It is a fast, fun and competitive game – one which has been researched little by the AI community. (Macleod 2004b) identifies Perudo as being an interesting game for AI search as, he argues, the brute force approaches used in Chess are unsuitable. Perudo is also preferred over Poker because it is less complex – which both makes it easier for a wide range of researchers to experiment with and because it makes the game less reliant on understanding of complex rules and more on "normal 'human' behaviour" (Macleod 2004b, p.269), and additional arguments on the suitability of Perudo for AI research are presented in (Macleod 2004c).

(Macleod 2004c) briefly outlines an attempt to use a fuzzy-logic approach to developing an AI player for Perudo, though the implementation and results are not given in any great detail. (Macleod 2004a) presents a more thorough description of the statistics of Perudo. This last paper also gives considerably more information on the implementation of probability based robot players for the game. This last work is discussed further later in this paper.

My approach here is to develop a number of robot players which each use simple heuristics and no player modelling. These can then be used in future work as a suitable set of test opponents for any adaptive AI players. In the following sections I describe the game of Perudo and a cut-down version, Perudish. I then describe the development of a small number of heuristic player strategies for two-player Perudish, each of which is tested against the others and against human players. I conclude by looking at future directions and ways of extending the current work – both that presented here and also that by Macleod in his various papers.

Perudo and Perudish

Basic Rules of Perudo

There are a number of known variants of Perudo. Here I outline some of the basic rules of a fairly standard game. In a full game of Perudo a number of players (generally 2-6) each start with five dice. All players roll and check their dice in secret. Then players take turns to bid on the total pool of dice. Each bid is an estimate, guess, or bluff, that there will be at least N many dice of a given face value, D. Following each bid, the next player must either make a higher bid or call.

After a call all hands are revealed. If there are less than N dice of face value D then the bidding player loses, and has one dice removed. Else, the calling player loses, and has one dice removed. Any dice with a face value of '1' are considered wild – and count towards the total number of dice whatever D is.

To make a new bid, the new bidding player must either increase D while keeping N constant or increase the number, N, or raise both D and N. Thus, bidding can be raised from three 3's to four 2's or three 4's or four 4's, and so on.

Other Rules

Because 1's are wild, special rules are used for bidding on 1's as they will generally occur in numbers half those of the other dice types. Some variants also allow players to make another call if they feel that the current bid is actually correct – thus avoiding the need to raise or call. Other special rules can come into play when a player is reduced to just one dice. More

detailed descriptions of the rules, including some variants, can be found in (Macleod 2004b) and online at *www.perudo.org* and *www.playperudo.com*.

Perudish: Simplified Perudo

Perudo is poker like in allocating a hidden hand to every player and in having a turn based bidding system. Perudo is a simpler problem to attack in a number of ways, however. In poker hands are only revealed when players call at the end of a hand – though the hands of players who 'fold' are not revealed at all and even the winner's hand is only revealed when at least one other player remains in the bidding. In comparison it should be easier to build player models in Perudo as more data on player styles is revealed – dice held by all players are revealed when a call is made.

A further simplification in comparison to Poker is the removal of cash bids. Much of the gameplay in Poker is centred not on the cards that make up the hands but on the cash bidding which has only an indirect relation to the cards held by different players.

For this paper I simplify Perudo even more in an attempt to build a base understanding before tackling the full game – in a similar way to how research in poker has often used reduced versions of the game as a means to gain basic insights into the game (Billings et al. 2002).

Here I limit play to repeated single hand games between two players. In every hand players have five dice apiece and a random player opens bidding. Subsequent bids must either raise or call, as per the description in the basic rules. 1's are wild, but players may not bid on them – removing one small complication. As every game involves only a single hand it is not necessary to include rules that apply when a player has only one dice, nor is it necessary to keep track of the numbers of dice held – although our application framework is designed to do so for purposes of future extension. This reduced game I have named 'Perudish'.

Simple Heuristics

Heuristics are powerful mechanisms used widely in everyday life as well as in AI research (Gigerenzer et al. 1999), and are certainly part of "normal 'human' behaviour" (to borrow Macleod's words). Before attempting to build complex AI robots to play Perudo or Perudish it seems sensible to build and test the abilities of robots using simple heuristics.

One such rule of thumb used personally when playing is to assume that for every three dice in game held by other players that there is probably another dice of my best type (either natural or 'wild'). So to get the number of additional dice of my chosen type, I simply divide the number of dice held by all other players by three. Additionally, personal experience is that rounding up this result is fairly reasonable, and often enough results in a win. In the opening round of an N player game, where each other player has five dice, I will generally reckon on there being two additional dice of any given type per opponent.

Is this heuristic backed up by the inherent probabilities of the game? Consider a two-player game of Perudish and assume that I have a preferred dice type – say fives. For each die thrown by my opponent, there is a one in three chance that it will come up with either a five or a one – in which case it counts towards the total number of fives. Table 1 shows the probabilities of the different outcomes of interest in this case.

Number of dice of any type, d, held by opponent	***Probability of this Outcome***
0	32/243
1	80/243
2	80/243
3	40/243
4	10/243
5	1/243
Total:	243/243

Table 1. For any given dice type (excluding 1's), the distribution of probabilities governing how many will be held by an opponent holding five dice. 1's are wild, reducing the number of different combinations of interest to 3 to the power of 5: 243.

Is my heuristic supported by this analysis? The likelihood that an opponent has two *or more* fives is 131 in 243. It is more likely than not that the bid will win if I am called. If I have more fives than any other face value, then the odds also would not appear to favour my opponent surviving being called if she raises in response to my bid. This suggests that it is quite reasonable to assume that an opponent holds two dice that match my best dice type – whatever that is.

A Very Simple Heuristic Player

Using this analysis, the desired behaviour of a simple heuristic player can be determined for the two player game of Perudish. After dice are rolled, select the favoured dice type, D_{BEST}, with the highest number of held dice, N_{HELD}. In the event of a tie, select the type with the highest face value. Add two to this to determine the highest bid supported, $N_{MAX\text{-}BID}$.

When responding to a bid, if the bid is lower than $N_{MAX\text{-}BID}$ times D_{BEST} then raise to this. If the bid is higher then call. This leaves the question of how to respond when the bid is equal to our highest bid. In some variants, as mentioned above, a special call could be used to declare that the player thinks that the bid is exactly right – but this rule is not present in the current game. Instead this is resolved randomly. With equal likelihood the response will be to either call or make a raise by increasing N_{BID} by one. As this robot always bids its highest estimate, except where forced to raise or call, we name this strategy HEURISTIC_MAX.

To test this I built a simple Perudish game according to the description below.

JPerudish

A simple Perudish game was implemented in Java: JPerudish. One of my goals was specifically to make the addition of new Perudish robot players as simple as possible, and so a Perudish Player interface class was defined. Any robot player can implement this interface, and can then be added to JPerudish with minimal effort. Currently it only requires a single change to one line of the Perudish code to add a new player once it has been written – with some minor modification it will be possible to remove even this small requirement.

Upon starting JPerudish, a list of players is presented – this includes a human player UI along with the different robot AI players. An experimenter can simply select any pair of players from those available and then start a game of 1, 10, 100 or 1000 hands (see Figure 1). As well as the robot players, a class has been implemented to allow human players to play against one another or against any of the robot players (see Figure 2). The game has been implemented to only reveal the dice held by robot players once a hand has been called. However, as all code runs on a single client machine, games between two human players are currently impractical. This could be easily fixed by supporting play over a network, although this is not currently a priority.

The Java interface for players has been designed to (hopefully) make the development of addition AI robot players as simple as possible. After any bid by any player, all players are notified of who bid and what they bid. After a call is made, all players are notified of the calling players and are shown the hands held by all other players. This information need not be used by robot players – indeed, is not used at all by the fixed heuristic players detailed in this report. Currently only the adaptive heuristic robot makes any use of this.

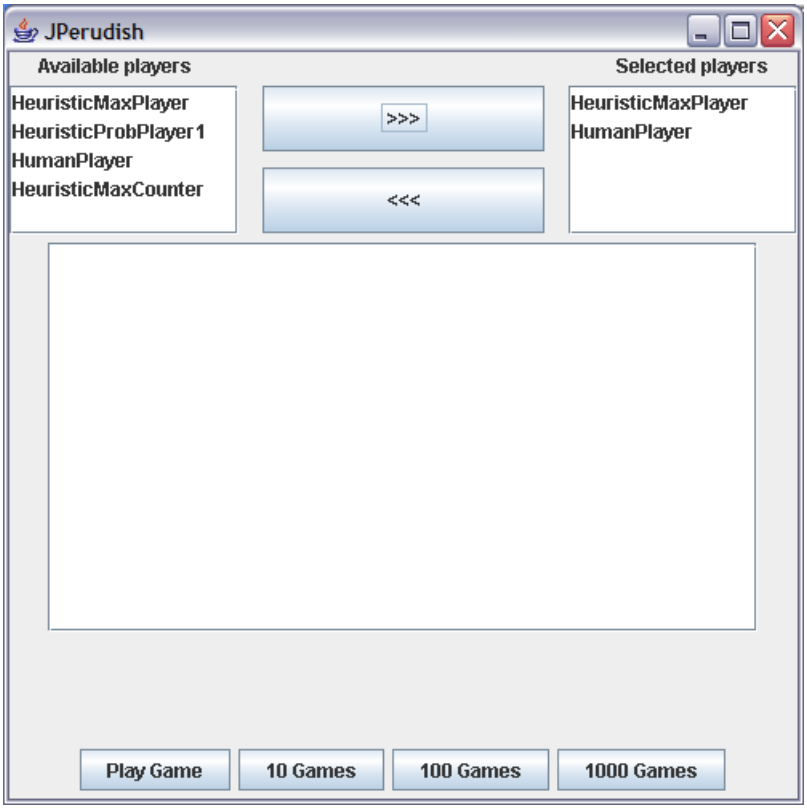

Figure 1. Main User Interface. The large text-area in the centre of the window is used to report the bidding and outcome of each game. Dice held are only revealed after a player calls.

Figure 2. HumanPlayer User Interface. Players can see all bids, but dice held by other players are not shown until a player calls.

While JPerudish is by no means of a commercial standard, code is available on request from the author. Should network play be added, it is hoped that online play over the internet will be made available, with suitable client-server modifications to the interface.

A Counter Strategy Robot Player

Although the previous analysis seemed to support the rules stated for the heuristic player, in practice it performs poorly against human opponents. In a number of tests it became apparent that human players could, after quite a small number of hands, work out the strategy being used by the computer. Once the strategy was understood the human player could then proceed to beat the computer in the majority of games. For example, while writing this I played twenty hands against the simple heuristic player, winning sixteen.

```
1  If holding < 2 dice of last bid type, then CALL
2  Else If holding > 2 dice of last bid type, then
         BID (last bid number+1) of lastbid type
3  Else If last bid type < my best type, then
         BID (last bid number) of my best type
4  Else if last bid type > my best type, then
         BID (last bid number+1) of my best type
5  Else If last bid type < 6, then
         BID (last bid.number) of (last bid type+1)
6  Else BID (last bid number+1) of my best type
```

Table 2. The counter strategy to the initial simple heuristic strategy always opens with a low bid. To respond to a previous bid the algorithm given here is used, either calling or making a small raise, as described below.

The obvious weakness is that players can use the simple heuristic player's bid to work out quite reliably what its N_{HELD} is. Indeed, in trials, human players unfamiliar with the robot strategy were able to determine what it was quite rapidly. Using this information the human player should be able to win any hand where their N_{HELD} of the computer's D_{BEST} is either less than *or* more than exactly two dice of that type. Thus, the odds dictate that a human player exploiting this weakness can win 67% percent of the time, after a computer bid. The human player can further pick up additional victories in some cases where they do hold exactly two dice of the type bid by the robot. These can occur when the player's D_{BEST} is higher or their greatest N_{HELD} is larger than the robot's. All in all this gives a considerable advantage to human players able to determine that the computer is playing the simple heuristic strategy.

This advantage can be further demonstrated by developing a counter strategy to HEURISTIC_MAX, which we name HMAX_COUNTER. When opening the bidding, HMAX_COUNTER always opens with a very low bid (one dice of face value two). Thus, which ever player starts the bidding, HMAX_COUNTER will have to respond to a bid from HEURISTIC_MAX. To do so, the algorithm given in Table 2 is used. In a game between these two strategies the HEURISTIC_MAX player will always be given the opportunity to make its maximum bid, giving the counter an opportunity to respond. If the counter strategy player has more or less than two dice of type that has been bid on it can call or raise appropriately and be guaranteed to win (steps 1 and 2). If the counter player has exactly two dice of the type bet on by the HEURISTIC_MAX player but has a different favoured dice type then the appropriate response is to change bidding to that type with the smallest raise possible (steps 3 and 4). If execution passes step 4 then the counter strategy will have lost in a game against HEURISTIC_MAX as both players have the same D_{BEST} and the bid made by the simple heuristic strategy matches exactly the best possible supported bid in the game. Nevertheless, some action needs to be taken, and steps 5 and 6 represent the smallest raises possible.

The success of the counter strategy was tested by playing the two strategies against one another for 10 games, each of 1000 hands. The Counter strategy won on average 757.7 hands per game, with a standard deviation of 16.1. While clearly superior in matches against our first heuristic player, this strategy remains weak against human player as it makes the assumption that its opponent is playing HEURISTIC_MAX and accordingly may call too readily, or unwisely raise, when playing a different opponent.

	vs. HEURISTIC_MAX	vs. HMAX_COUNTER
HMAX_RANDOM wins	496	548.75
Standard Deviation	16.9	14.0

Table 3. HMAX_RANDOM provides a greater challenge for human opponents, but does not perform particularly well against the original heuristic strategy. Wins averaged over 20 games each of 1000 hands.

Indeed, where two HMAX_COUNTER players are pitted against each other a race condition exists. If both hands contain two of more of some common type, then once a bid on that type has been placed subsequent bids will continuously raise and re-raise without end (caused by step 2 of the algorithm in Table 2). A simple test inserted before step 2 to ensure that the bid is feasible given the number of dice in play can prevent this.

A More Challenging Heuristic

Returning to the original robot player, the flaw with the strategy is not that it makes bad estimates. Rather the flaw is that its bidding strategy always reveals what it is holding. An obvious solution is to open with a random bid below the highest estimate. In response to a bid by another player, instead of raising immediately to the best estimate the player should again raise randomly towards the maximum bid. Such a player can be much more challenging as it is no longer possible to determine what they are holding. A version of this strategy was implemented where the number of dice bid in an opening bid was selected with uniform chance from the range $[1..N_{MAX\text{-}BID}]$ and the dice type similarly selected from the range $[2..D_{BEST}]$. For a bid made in response to a bid by another player, the minimum values in these ranges were set to correspond to the bid received – and every response being either a valid raise or a call.

This strategy, HMAX_RANDOM, can perform quite well against human players, although this was not deliberate. This is due to the very random nature of bidding making the player less predictable – but still beatable by players who understand its strategy. In thirty games, played while writing this sectiion, I defeated it 21 games to 9. It is however a very unnatural style of play, and one which might not work as well in the multiplayer game.

The performance of this strategy against the previous two is detailed in Table 3.

Unsurprisingly, HMAX_RANDOM defeats HMAX_COUNTER – although not by a great margin. The assumptions built into HMAX_COUNTER are not valid here, and it is to be expected that it will lose more often. HMAX_RANDOM versus HEURISTIC_MAX results in quite even performance – while HMAX_RANDOM starts low and bids up to its limit, both strategies use the same algorithm for determining their bidding limits and will call once the limit is exceeded.

Opponent Modelling

In Perudo human players commonly use the bids of other players as a means of attempting to gauge what dice are held by those players – and awareness of this gives rise to bluffs and attempts at deception. Including these aspects in play clearly requires some form of opponent modelling as without this players only have access to their own dice and information on statistical likelihoods.

(Macleod 2004a) includes some degree of opponent modelling in a scheme for implementing computer players for Perudo. Four key parameters are used to characterise each computer player, supported by individually held tables of probability distributions for the dice likely held by opponents in each hand. In principle this element of the system works in a similar, though simpler, fashion to some of the work on opponent modelling in poker. There tables of probabilities are built to represent the likely hands held by players given that they remain in play as bidding increases {as detailed in \Davidson, 2000 #112}.

The first two parameters, of the system described in (Macleod 2004a), P_{MIN} and P_{MAX} limit the bids made by players according to how likely it is that the bids will survive being called. A player will not make any bid that has less than P_{MIN} chance of surviving being called, and when forced to make a bid this risky it will always call instead. Another parameter, P_{TRUTH}, stores for each player the probability that they will assume that a bid on a certain type of dice means that the player making that bid does indeed have more dice of that type than would be expected on average. On 'believing' a bid, the player then increases its estimate of how many dice of that type exist by adjusting its probability distribution table. The final weighting parameter quantifies the scale of adjustment made to the tables each time a bid is believed.

One key observation here is that in a game with multiple players it might make more sense to keep separate P_{TRUTH} values for each opponent. Each 'truth' parameter should be modified based on how often the corresponding opponent makes an honest or dishonest bid. Instead, when learning is enabled, each player appears to record only a single P_{TRUTH} value. In this scheme it appears to be assumed that all players are as truthful as each other.

I feel, as does Macleod, that successful opponent modelling is key to success in Perudo. There is clearly scope for extending the current work, and this will be one focus of my follow on work, the beginnings of which are detailed next.

A Simple Adaptive Heuristic Player

	vs. HEURISTIC_MAX	vs. HMAX_COUNTER	Vs. HMAX_RANDOM
HEURISTIC_ADAPTIVE wins	660	758	526

Table 4. HEURISTIC_ADAPTIVE performs very well against the fixed strategies, but less well against the strategy with a random component in play. These results are from single games of 1000 hands.

While the goal of this work is to develop a number of competitive AI robot players for Perudo, a preliminary step is to demonstrate that a simple robot player can successfully adapt to play well against different opponents. Our first adaptive robot player is certainly very simple. Using the HEURISTIC_COUNTER strategy as a start point, variables to track two aspects of an opponents strategy are added. One variable is used to keep track of whether an opponent tends to bid on the highest dice type which they hold most of – bids_on_type. The second variable keeps trach of whether the opponent tends to open with low bids compared to what they might expect to be revealed given their hand – opens_low. Both of these variables are given values in the range 0 to 1, and are intended to reflect the probability of an opponent taking the corresponding action. Both are initially set to 0.5.

For example, if the adaptive robot is successful it might determine that an aggressive bluffing opponent has a low value for bids_on_type and a low value for opens_low. An aggressive non-bluffing player might be given a low value for opens_low but a large one for bids_on_type. More conservative players would have higher values for opens_low.

To implement this, the adaptive robot reviews the bids made when hands are revealed to determine if opponents opened low and whether they tended to bid on their favoured dice. Values of these variables are then adjusted up or down accordingly – plus or minus 0.1 applied to opens_low and plus or minus 0.05 to bids_on_type for each bid made in a single hand.

These values are then used in play in the following way. If it is known that an opponent tends not to open low (opens_low < 0.3) and the adaptive player has fewer than two dice of the type just bid in an opponents opening bid, then it will call. This should defeat aggressive play if the adaptive players hand does not support the previous bid. If the opponent does open low (opens_low > 0.7) then raise an opening bid without calling, either maintaining same dice type (bids_on_type > 0.5) or change to favoured dice type. This should prevent losing against cautious play.

If still undecided, or if not responding to an opening bid, the following steps are followed. If opponent generally bids on their favoured type (bids_on_type > 0.7), then only call them if the number of dice bid is significantly higher than the number held (currently set to call if it exceeds the number held by 4 or 5 in a two player game of five dice each). If opponent does not bid on their favoured type (bids_on_type < 0.3), and the adaptive player holds less than two dice of the type just bid then it will call.

Finally, if the response is still not determined, the adaptive robot drops into the list of responses used by HEURISTIC_COUNTER to ensure that some (reasonable) response is always made. The performance of this adaptive robot against the other robots described in this paper is shown in Table 4.

The HEURISTIC_ADAPTIVE player learns very clearly how to play well against the original and counter strategies. Success against the less predictable HMAX_RANDOM strategy is not far above 50%, showing that it struggles learning how to play well against this (quite random) opponent. It does, however, demonstrate that opponent modelling can be incorporated into AI for Perudo. Indeed, it is unlikely that many human players play as randomly as HMAX_RANDOM does.

Future Work

This paper reports on quite early work on developing AI players for Perudo. Plans exist for extending this work in a number of ways, some of which are detailed below.

N-Player Perudish

When playing Perudo or Perudish with more than two players a number of other issues affect successful play, and some of these are mentioned in (Macleod 2004a). One key observation is that a player cannot lose as long as it neither calls nor is called by another player. But if bids get so high as to be extremely unfeasible then a player is better to call than raise the bidding yet further and subsequently lose a dice by being called by the next player. Thus one way to succeed is to make low bids that are unlikely to be called. But as bids monotonically rise as bidding proceeds in a circular fashion around all players, the bids should not be so low as to allow bidding in the current hand to return to the player. Additional dynamics can arise from attempts to ‘get’ specific non-adjacent players, by making helpful, low, bids. Such circumstances arise in

human play when, for example, some players hold all of their starting dice while the majority have lost one or more dice each.

We suggest that for N-Player Perudish the number of players should be between 2 and 6, and each player should start with five dice. Ones remain wild throughout, but cannot be bid on. The losing player after each call loses a dice and then – unless out of the game – opens the bidding for the next hand. These changes make the game significantly more similar to Perudo but still somewhat simpler, and make it possible to explore richer game dynamics.

AI Players

The most obvious extension to this work is to develop more interesting AI players. A range of approaches are possible, focussing on the probabilities inherent in the game and/or extending the opponent models. Some significant changes to the current implementation will be required as the game is generalised to the n-player game where dice are lost by the loser in each hand, and this is likely to be a first step in improving and extending the current AI.

Conclusions

In this paper I have reviewed current work on AI for the game of Perudo. A simplified game, Perudish, was introduced and a range of fixed and adaptive robot players were developed for the 2-player game. Performance of these against each other – and in some cases against human players – has been reviewed. These is currently some need to perform more testing against human opponents, particularly for the HEURISTIC_ADAPTIVE robot, as well as development of new and better AI players.

The paper has shown that opponent modelling in Perudish can lead to some success in play and that it should be a productive direction for further exploration.

Acknowledgements

For useful discussions and advice on developing this work I would like to thank Colin Fyfe, Padraig Cunningham and the members of the TCD Machine Learning Group, and Sarah Jane-Delaney of DIT. I would also like to thank the Carnegie Trust for the Universities of Scotland for supporting this research.

References

Billings, D., A. Davidson, J. Schaeffer and D. Szafron (2002). "The Challenge of Poker." *Artificial Intelligence Journal* **134**(1-2): 201-240.

Davidson, A., D. Billings, J. Schaeffer and D. Szafron (2000). Improved Opponent Modeling in Poker. *International Conference on Artificial Intelligence (ICAI'2000)*, 1467-1473.

Gigerenzer, G., P. M. Todd and t. A. R. Group (1999). *Simple Heuristics That Make Us Smart*. Oxford, Oxford University Press.

Macleod, A. (2004a). A Novel Platform for Developing Mundane Skills in Artificial Players. *Game'On*, 8th-10th Nov., Ghent.

Macleod, A. (2004b). Perudo as a Development Platform for Artificial Intelligence. *CGAIDE*, Q. Mehdi and N. Gough (eds.), 8th-10th Nov., Microsoft Campus, Reading, UK.

Macleod, A. (2004c). Selecting Games for Artificial Intelligence Research. *The Second Annual International Workshop in Computer Game Design and Technology*, 15th-16th Nov., Liverpool.