Optimal, Approximately Optimal, and Fair Play of the Fowl Play Card Game

Todd W. Neller, Marcin Malec, Clifton G. M. Presser, and Forrest Jacobs

Gettysburg College, Dept. of Computer Science, Gettysburg, Pennsylvania, 17325, USA,
tneller@gettysburg.edu, khazum@gmail.com,
cpresser@gettysburg.edu, forrestjacobs@gmail.com
http://cs.gettysburg.edu/~tneller

Abstract. After introducing the jeopardy card game Fowl Play, we present equations for optimal two-player play, describe their solution with a variant of value iteration, and visualize the optimal play policy. Next, we discuss the approximation of optimal play and note that neural network learning can achieve a win rate within 1% of optimal play yet with a 5-orders-of-magnitude reduction in memory requirements. Optimal komi (i.e. compensation points) are computed for the two-player games of Pig and Fowl Play. Finally, we make use of such komi computations in order to redesign Fowl Play for two-player fairness, creating the game Red Light.

Keywords: Fowl Play, jeopardy, card game, value iteration, neural networks, komi, Pig, Red Light

Introduction

Designed by Robert Bushnell and published by Gamewright in 2002, Fowl Play™ is a jeopardy card game variant of the folk jeopardy dice game Pig [1–3]. The object of Fowl Play is to be the first of 2 or more players to reach a given goal score. We focus here on the 2 player game where the goal score is 50 points. A shuffled 48-card deck contains cards of two types: 42 chickens and 6 wolves. Each turn, a player repeatedly draws cards until either a wolf is drawn or the player holds and scores the number of chickens drawn, i.e. the *turn total*. During a player's turn, after the first required draw[1], the player is faced with two choices: *draw* or *hold*. If the player draws a wolf, the player scores nothing and it becomes the opponent's turn. If the player draws a chicken, the player's turn total is incremented and the player's turn continues. If the player instead chooses to hold, the turn total is added to the player's score and it becomes the opponent's turn.

[1] Although a turn's initial draw requirement is not clearly or explicitly stated, the rules seem to imply the requirement, and it is necessary to avoid stalemate. Although the rules state "You can stop counting and collect your points at any time as long as you don't turn over a wolf!", there is an implication that one has *started* counting chickens/points. Consider the scenario where players have scores tied at 49 and the deck contains a single wolf card. It is in neither player's interest to draw the wolf card, so rational players would infinitely hold as a first action if permitted.

All cards drawn are discarded after a player's turn. If the last, sixth wolf is drawn, the entire deck is reshuffled before the next turn begins.

For such a simple card game, one might expect a simple optimal strategy as with blackjack (e.g., "Stand on 17," etc.). As we shall see, this simple card game yields a much more complex and intriguing optimal policy, computed and presented here for the first time.

We begin by presenting play that maximizes expected score per turn, explaining why this differs from optimal play. After describing the equations of optimal play and the technique used to solve them, we present visualizations of such play and compare it to optimal play of a simpler, related dice game Pig. Next, we turn our attention to the use of function approximation in order to demonstrate the feasibility of a good, memory-efficient approximation of optimal play. Finally, we apply our analytical techniques towards the parameterization, tuning, and improved redesign of the game with komi for optimal fairness.

Maximizing Score

The game of Fowl Play is simple to describe, but is it simple to play well? More specifically, how can we play the game optimally? This, of course, depends on how we define "optimal". As a first step, let us suppose that a player wishes to maximize score gain per turn.

Let us begin by defining relevant variables to describe each game state:

– i - current player score
– j - opponent player score
– k - turn total
– w - wolves drawn since last shuffle
– c - chickens drawn since last shuffle

We assume that a player with a turn total sufficient to win will hold. Thus, for non-terminal states, $i + k < g$, where g is the goal score of 50. Also, one cannot have drawn more chickens on one's turn than have been drawn since the last deck shuffle, so $k \leq c$.

Further, let $w_{\text{total}} = 6$ and $c_{\text{total}} = 42$ denote the total number of wolf and chicken cards, respectively. Let $w_{\text{rem}} = w_{\text{total}} - w$ and $c_{\text{rem}} = c_{\text{total}} - c$ be the remaining number of wolf and chicken cards in the deck, respectively.

In order to maximize score, each decision to draw should be expected, on average, to increase the turn score. The expected gain should exceed the expected loss. The expected gain from a draw is the probability of drawing a chicken times the turn total gain of 1: $\frac{c_{\text{rem}}}{w_{\text{rem}}+c_{\text{rem}}}$. The expected loss from a draw is the probability of drawing a wolf times the turn total itself: $\frac{w_{\text{rem}}}{w_{\text{rem}}+c_{\text{rem}}}k$. Thus, a player maximizing score draws a card as required at the beginning of a turn and when the expected gain exceeds the loss:

$$\frac{c_{\text{rem}}}{w_{\text{rem}} + c_{\text{rem}}} > \frac{w_{\text{rem}}}{w_{\text{rem}} + c_{\text{rem}}}k$$

Simplifying, this condition is equivalent to:

$$c_{\text{rem}} > w_{\text{rem}}k$$

So the player maximizing expected score opts to draw a card when the number of chickens remaining exceeds the number of wolves remaining times the turn total.

However, there are many circumstances in which one should deviate from this score-maximizing policy. *Risking points is not the same as risking the probability of winning*. Put another way, playing to maximize points for a single turn is different from playing to win.

To illustrate this point, consider the following example. Suppose your opponent has 49 points. There are 2 remaining cards in the deck: 1 wolf and 1 chicken. You have a score of 47 and a turn total of 2. Drawing would give an expected score gain of $\frac{1}{2}$ and a greater expected score loss of 1, so the score-maximizing player would hold.

However, the player playing to maximize the probability of winning will draw a card in this situation. If a chicken is drawn with probability $\frac{1}{2}$, the player then holds and wins, so the probability of winning with a draw is at least $\frac{1}{2}$. In fact, the probability is greater than $\frac{1}{2}$, as a draw of a wolf would result in a reshuffle and a non-zero probability that the opponent will draw a wolf on their next turn, allowing for possibility of the player winning in some future turn.

Compare this with a decision to instead hold with 2 points. The opponent begins the turn with a tied 49-49 score and the same deck situation. By the same reasoning, the opponent has a greater than $\frac{1}{2}$ probability of winning, so in choosing to hold, the player would be choosing a probability of winning strictly less than $\frac{1}{2}$ and thus strictly less than the probability of winning by choosing to draw.

In the next section, we form the equations that describe optimal play, and show how how it significantly deviates from and outperforms score-maximization play.

Maximizing the Probability of Winning

Let $P_{i,j,k,w,c}$ be the player's probability of winning if the player's score is i, the opponent's score is j, the player's turn total is k, and the wolf and chicken cards drawn since the last shuffle are w and c, respectively. In the case where $i + k \geq 50$, $P_{i,j,k,w,c} = 1$ because the player can simply hold and win. In the general case where $0 \leq i, j < 50$ and $k < 50 - i$, the probability of an optimal player winning is

$$P_{i,j,k,w,c} = \begin{cases} P_{i,j,k,w,c,\text{draw}} & k = 0, \text{and} \\ \max\left(P_{i,j,k,w,c,\text{draw}}, P_{i,j,k,w,c,\text{hold}}\right) & \text{otherwise.} \end{cases}$$

where $P_{i,j,k,w,c,\text{draw}}$ and $P_{i,j,k,w,c,\text{hold}}$ are the probabilities of winning if one draws and holds, respectively. These probabilities are given by:

$$\begin{aligned} P_{i,j,k,w,c,\text{draw}} = & \tfrac{c_{\text{rem}}}{w_{\text{rem}}+c_{\text{rem}}} P_{i,j,k+1,w,c+1} \\ & + \tfrac{w_{\text{rem}}}{w_{\text{rem}}+c_{\text{rem}}} \begin{cases} 1 - P_{j,i,0,w+1,c} & w < w_{\text{total}} - 1, and \\ 1 - P_{j,i,0,0,0} & \text{otherwise.} \end{cases} \end{aligned}$$

$$P_{i,j,k,w,c,\text{hold}} = 1 - P_{j,i+k,0,w,c}$$

The probability of winning after drawing a wolf or holding is the probability that the other player will not win beginning with the next turn. The probability of winning after drawing a chicken depends on the probability of winning with incremented turn total and chickens drawn.

At this point, we can see what needs to be done to compute the optimal policy for play. If we can solve for all probabilities of winning in all possible game states, we need only compare $P_{i,j,k,w,c,\mathrm{draw}}$ with $P_{i,j,k,w,c,\mathrm{hold}}$ for our current state and draw or hold depending on which gives us a higher probability of winning.

Solving for the probability of a win in all states is not trivial, as dependencies between variables are cyclic. Beginning with a full deck, two players that manage to draw a wolf at the end of the next 6 turns will find themselves back in the exact same game state. Put another way, game states can repeat, so we cannot simply evaluate state win probabilities from the end of the game backwards to the beginning.

Solving with Value Iteration

Value iteration [4–6] is an algorithm that iteratively improves estimates of the value of being in each state until such estimate updates reach a terminal condition. We can summarize the value iteration algorithm of [4] by saying that state values are computed from Bellman's optimality equations by iteratively computing the right-hand-side equation expressions using the previous (initially arbitrary) value estimates, and assigning these new estimates to the left-hand-side equation state value variables. With each iteration, one tracks the maximum magnitude of change to any state value estimate. When the largest estimate change in an iteration falls below a desired threshold, the estimates have sufficiently converged and the algorithm halts.

In our application of value iteration as in [1], there are only two points to be made. The first is that each state value is defined to be the probability of winning with optimal play from that state forward. Winning and losing terminal states thus have state values of 1 and 0, repectively. The second point is that we generalize Bellman's equations to model a two player non-drawing game as described in our previous optimality equations.

A similar technique is employed to evaluate relative player strengths. Given a fixed play policy for two players, we form equations to compute each player's probability of winning in all possible non-terminal states as in [3].

The score-maximizing player ("max-score player") makes the same decision as the optimal player in 90.04% of the 10,487,700 non-terminal game states. As a result, the max-score player wins with probability $.459$ against the optimal player (probability $.484$ as first player and $.434$ as second player).

Visualizing the Solution

We can think of the optimal policy as being a boundary between draw and hold conditions in the 5-dimensional state space, yet this is not easily visualized. What we do to give the reader the best sense of optimal play is to present a selection of 3-dimensional graphs where 2 of the 5 dimensions, w and c, have been fixed. See Figure 1, where $w = c = 0$ at the beginning of the turn. This is the situation both at the beginning of the game and when the deck has been reshuffled after the 6th wolf has been drawn.

For the given initial values of w and c at the beginning of the turn, the vertical height of the surface represents the number of cards a player should draw before holding for every possible pair of player and opponent scores.

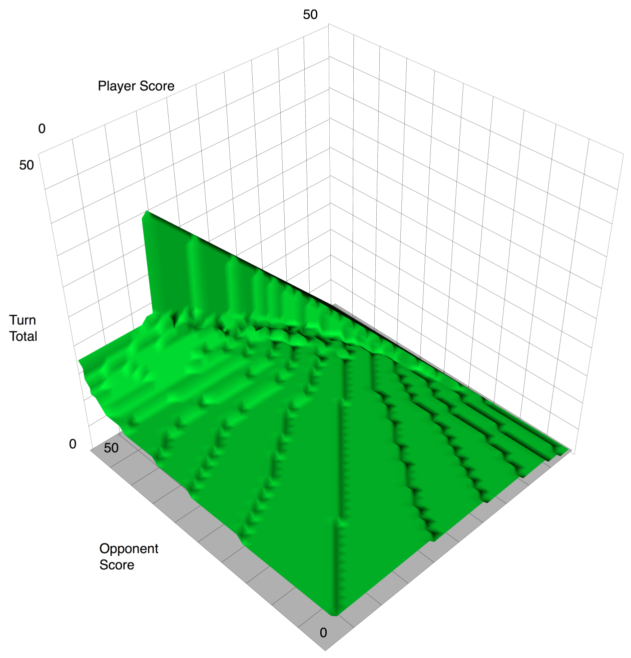

Fig. 1. The draw/hold surface for the optimal Fowl Play player when no cards have been drawn.

As one varies the number of chickens and wolves drawn before the beginning of the turn, one can see that a player's play varies considerably and especially at score extremes (see Figure 3).

Fowl Play may be thought of as a card game variation of the folk jeopardy dice game Pig [1]. In the game of Pig, the first player to reach 100 points wins. Each turn, a player rolls a 6-sided die until either (1) the player holds and scores the sum of the rolls, or (2) rolls a 1 ("a pig") and scores nothing for the turn.

Fowl Play has many similarities to Pig. Both games have a similar first-player advantage in 2-player optimal play with the probability of the first of two optimal players winning being 53.06% and 52.42% for Pig and FowlPlay, respectively. Both games have a similar number of expected actions for an optimal two-player game with 167.29 and 164.98 expected player actions for Pig and Fowl Play respectively. One can see some similarity in general hold boundaries, with the player ahead/behind taking lesser/greater risks.

However, Fowl Play has important differences as well. In Pig, there is no limit to the potential score of a turn (see Figure 2), whereas in Fowl Play, the score is necessarily limited to the number of chicken cards remaining in the deck. The shared deck intro-

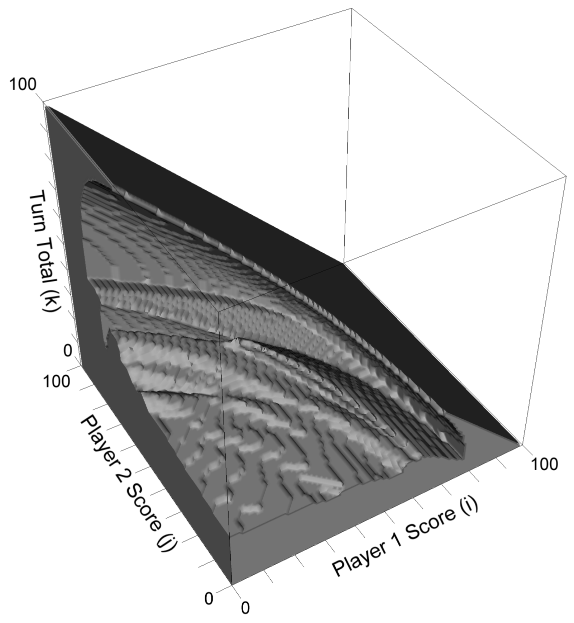

Fig. 2. The roll/hold surface for the optimal Pig player.

duces interesting dynamics to play, as the outcome of one's turn determines the initial situation and scoring potentials for the opponent's next turn.

Of particular interest are situations where it becomes rational to draw a wolf after all chickens have been drawn. There exist situations where, with few chickens and an even number of wolves remaining, where a player drawing the last chicken should deliberately draw one of the remaining wolves, forcing a chain of wolf draws that leaves the player with a fresh deck and a greater upside for scoring.

Whereas simple mental arithmetic suffices to closely approximate optimal play for Pig, we have yet to find sufficient features that would allow mental arithmetic to closely approximate optimal play for Fowl Play.

Approximating Optimal Play

Having computed optimal player for 2-player Fowl Play, we observe that the computation required the storage of 10,487,700 floating point numbers. The policy need not be represented as 10,487,700 boolean draw/hold values, however. We may simply map each (i, j, w, c) tuple to the *hold value*, the minimum turn total k for which the player holds. This reduces space requirements for storing the optimal policy to $50 \times 50 \times 6 \times 43 =$ 645,000 integers.

However, this is still a large memory requirement for modern mobile and embedded systems. As an assignment for an upper-level elective course on advanced game artificial intelligence, students were provided the optimal policy for Fowl Play and chal-

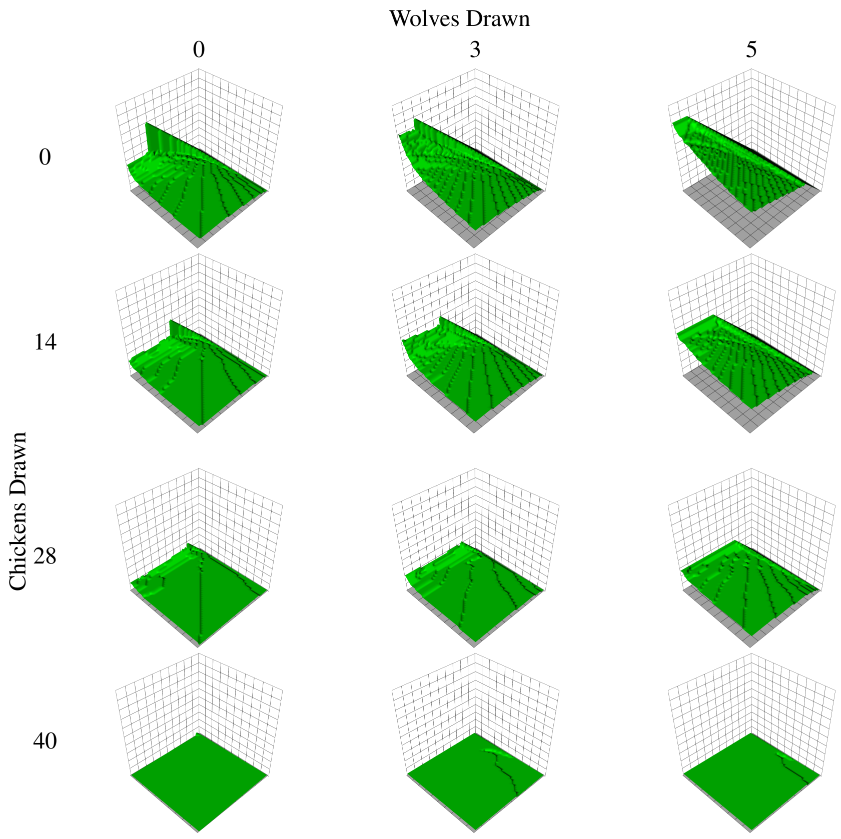

Fig. 3. The draw/hold surface with varied numbers of wolves and chickens drawn from the deck. The axis of the graphs are defined as the previous figure.

lenged to create a function approximation that achieves near-optimal play with a very small memory requirement.

The reader should note that there are a number of possible design choices for the function approximation task. One can:

- Approximate $V(i,j,k,w,c)$, the state value function equivalent to the expected win probability with optimal play. Then using the optimality equations, one can determine which next action maximizes the function.
- Approximate $Q(i,j,k,w,c,a)$, the action value function, where $a \in \{\text{draw}, \text{hold}\}$. Note that $\max_a Q(i,j,k,w,c,a) = V(i,j,k,w,c)$.
- Approximate the policy function $\pi(i,j,k,w,c)$ that maps to *draw* (1) or *hold* (0). The problem then becomes a classification problem, in that one seeks to classify states as *draw* or *hold* states.
- Approximate a function $\text{draws}(i,j,w,c)$ derived from $\pi(i,j,k,w,c)$ that maps the state at the beginning of the turn (when the turn total $k = 0$) to the total number of *draw* actions taken before a *hold* action. This is the minimum k for which $\pi(i,j,k,w,c+k)$ maps to *hold*.

Two students independently demonstrated that multi-layer feed-forward neural networks showed the greatest promise for closely approximating optimal play with minimal memory requirements. Since that time, the approach has been refined and further improved. We here present the details of the best approach to date.

Network Input: In addition to the input features i, j, k, w_{rem}, and c_{rem}, the features $\frac{c_{\text{rem}}}{w_{\text{rem}}+c_{\text{rem}}}$ and $\frac{w_{\text{rem}}}{w_{\text{rem}}+c_{\text{rem}}} \times k$, the probability of drawing chicken and wolf cards, respectively, aided in the function approximation. These seven features scaled to range $[-1, 1]$ form the input to the network.

Network Output: The single output unit classifies *draw* or *hold* actions according to whether the output is above or below 0.5, respectively. For training purposes, target outputs are 1 and 0 for *draw* and *hold*, respectively.

Network Architecture: Optimizing the number of hidden units empirically so as to boost learning rate and generalization, the single hidden layer worked best with 13 units. Thus, with an additional input and hidden layer bias unit, there are $8 \times 13 = 104$ weights from the input layer to the hidden layer, and 14 weights from the hidden layer to the single output unit, for a total of 118 weights. The activation function for all hidden and output units is the logistic function $f(x) = \frac{1}{1+e^{-x}}$.

Network Training: Network weights are initialized using the Nguyen-Widrow algorithm [7]. State inputs are generated through simulations of games with optimal play so as to focus learning on the most relevant parts of the state space. For each non-terminal decision state input, the network output is computed. Optimal play is then referenced, and error is computed as the difference between 1 (*draw*) or 0 (*hold*) and the network output. Standard backpropagation [8] is then applied with learning rate $\alpha = 0.1$ to update network weights.

Online learning occurs for a duration of 100,000 games. Performance is then evaluated through network play against an optimal player for 1,000,000 games. If the win rate is better than 49.5% and the highest win rate so far, we re-evaluate using policy iteration with $\epsilon = 1 \times 10^{-6}$. We then continue to alternate between training and evaluation for total of 400 iterations.

The network resulting from such training achieves an average win rate of $49.58 \pm 0.10\%$ against an optimal player over 180 re-runs with a highest win rate of 49.79%. This represents an impressive 5-orders-of-magnitude compression of optimal play policy information with very little loss of play quality.

Optimal Komi and the Game of Red Light

Given that computation of optimal play is feasible for Fowl Play, we now turn our attention towards further analysis of the game, and parameterization of the game in order to tune the game for fairness and length.

Pig and Fowl Play both have a significant first player advantage. In fact, we can compute optimal komi[2], i.e. compensation points, for the second player such that the first player win probability most closely approximates $\frac{1}{2}$ between optimal players. At the same time, we can also compute the expected number of game actions as in indicator of expected game length.

For example, the game of Pig played between two optimal players will result in a first player wins 53.06% of games after 167.29 expected player actions. However, if the second player starts with a komi of 4 points, the first player wins 50.16% of games after 164.01 expected player actions.

With optimal Fowl Play, the first player wins 52.42% of games after 164.98 expected player actions. However, if the second player starts with a komi of 1 point, the first player wins 50.54% of games after 163.23 expected player actions.

Just as the commercial card game Fowl Play derives from the folk dice game Pig, we next apply our analytical methods to redesign Fowl Play as a new jeopardy game we call Red Light.

Red Light

Red Light is a jeopardy game requiring red poker chips ("red lights"), green poker chips ("green lights")[3], and a bag or container that allows players to shuffle and randomly draw concealed chips.

For a 2 player game, we suggest 4 red chips, 24 green chips, a goal score of 50, and a non-starting player initial score of 1.

The first player to reach the goal score wins. The starting player is determined by fair means (e.g. coin toss, alternation, etc.) A player begins a turn by drawing a poker chip from the bag or container.

If the drawn chip is a green light, the player sets it aside in a "turn total" pile. After drawing, the player then decides whether to draw again or hold. If the player holds, the number of green lights in the turn total pile are scored, the chips are placed in a discard pile, and play passes to the next player.

If the drawn chip is a red light, the player scores nothing for the turn, places all chips in the discard pile, and play passes to the next player. If the red chip was the last,

[2] Komi is a Japanese Go term, short for "komidashi".

[3] For players with red/green color-blindness, we recommend use of yellow or light green chips for sufficient contrast.

fourth red light, discarded chips are returned to the bag or container and shuffled before the next turn.

The number of red and green chips for Red Light was chosen by varying the number of chips in play, analyzing the game, and optimizing komi. We found an extraordinarily good approximation of a fair game for 4 red chips and 28 chips total with a komi of 1, deviating from a fair win probability by 0.00001. With larger numbers of total chips, the game length shortens, and the ideal komi increases. We have opted for smaller numbers of chips (< 50) in our redesign because poker chips are often sold in units of 25 or 50. The expected total number of player actions per game is 169.748, so Red Light has almost the same expected game length as Fowl Play.

Finally, we note that the 2-player Red Light can effectively be played with standard playing cards in the same manner as Fowl Play using Ace through 7 of each suit. Aces can signify red lights, while all other cards signify green lights.

Conclusion

In this paper, we have solved the jeopardy card game Fowl Play for the first time, demonstrating significant difference between score-maximizing and optimal play. We have visualized optimal play, highlighting similarities to and differences from its ancestor, the folk jeopardy dice game of Pig. Our work with neural networks has shown that it is possible to approximate optimal play well, reducing memory requirements by five orders of magnitude while only losing a fraction of a percent in win probability. Finally, we have parameterized the number of good and bad cards in the game as well as second-player komi in order to optimize the game for fairness. In this we succeeded, creating an almost perfectly fair[4] variant of the game we call Red Light.

There are interesting questions that yet remain: How well does optimal 2-player play approximate optimal 3-player play if one treats the two opponents as a single abstract opponent? What human-playable strategies exist that significantly improve upon max-score play? How well can people play against an optimal player?

We hope that other teachers and researchers find Fowl Play and/or Red Light of value in their work as well. They are excellent examples of games with minimal rules, yet complex optimal play.

References

1. Neller, T.W., Presser, C.G.: Optimal play of the dice game pig. The UMAP Journal **25**(1) (2004) 25–47
2. Neller, T.W., Presser, C.G.: Pigtail: A pig addendum. The UMAP Journal **26**(4) (2005) 443–458
3. Neller, T.W., Presser, C.G.: Practical play of the dice game pig. The UMAP Journal **31**(1) (2010) 5–19
4. Sutton, R.S., Barto, A.G.: Reinforcement Learning: an introduction. MIT Press, Cambridge, Massachusetts (1998)

[4] i.e. fair after first-player determination

5. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton, New Jersey, USA (1957)
6. Bertsekas, D.P.: Dynamic Programming: deterministic and stochastic models. Prentice-Hall, Upper Saddle River, New Jersey, USA (1987)
7. Nguyen, D., Widrow, B.: Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: Neural Networks, 1990., 1990 IJCNN International Joint Conference on. Volume 3. (June) 21–26
8. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, New York (1995)