PEDAGOGICAL POSSIBILITIES FOR THE DICE GAME PIG[*]

Todd W. Neller
Gettysburg College
tneller@gettysburg.edu

Clifton G.M. Presser
Gettysburg College
cpresser@gettysburg.edu

Ingrid Russell
University of Hartford
irussell@hartford.edu

Zdravko Markov
Central Connecticut State University
markovz@ccsu.edu

ABSTRACT

Simple examples are teaching treasures. Finding a concise, effective illustration is like finding a precious gem. When such an example is fun and intriguing, it is educational gold. In this paper, we share the jeopardy dice game of Pig, which has extremely simple rules, engaging play, and a complex optimal policy. We describe its historical uses in mathematics, and share ways in which we have used the game to teach basic concepts in CS1, and intermediate concepts in introductory artificial intelligence, networking, and scientific visualization courses. We also describe the rich challenges Pig offers for undergraduate research in machine learning.

INTRODUCTION

The jeopardy dice game of Pig has simple rules, yet has complex dynamics. Pig and its variants have found many pedagogical uses from middle school mathematics to introductory computer science to machine learning undergraduate research. Here we seek to collect and focus on the uses of Pig in an undergraduate computer science curriculum.

The Game of Pig

The object of Pig is to be the first player to reach 100 points. Each turn, a player repeatedly rolls a die until either a 1 is rolled or the player holds and scores the sum of the rolls (i.e., the *turn total*). At any time during a player's turn, the player is faced with

[*] Copyright © 2006 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

two choices: *roll* or *hold*. If the player rolls a 1, the player scores nothing and it becomes the opponent's turn. If the player rolls a number other than 1, the number is added to the player's turn total 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.

Pig is a folk jeopardy dice game with many variants and an interesting history [15]. The rules of Pig, as we describe them, are the earliest non-commercial rules found in the literature, described in 1945 by John Scarne [19]. Scarne also suggested some modifications to the folk game which we will not include here for simplicity, following the basic rules that also appear in [1, 6, 9]. Of these, Reiner Knizia wrote "The simplicity and purity of this game makes PIG ideal to demonstrate the tactical approach to jeopardy games." [9, p. 129].

PREVIOUS PIG PEDAGOGY

Teachers of Mathematics were quick to note the value of Pig as an entertaining example for teaching probabilistic concepts. Published teaching materials reveal many variants (e.g. 2 dice, special effects of doubles), although such exercises are often easily translated to our fundamental rules.

The most extensive curricular use of Pig is by the Interactive Mathematics Program, which uses The Game of Pig as the core example for its high school first year probability curriculum [8]. The core focus is on maximizing expected points for a single turn.

Students experiment and use area models of probability to explore tactics for Pig and some variants. Some middle school teaching materials (e.g. [4]) ask how many rolls a player should make before holding. This is an example of the *gambler's fallacy*. The Statistics strand of "Mathematics in the New Zealand Curriculum" does a particularly good job of empirically demonstrating the weakness of this approach, comparing "hold after *n* rolls" policies versus "hold at turn total *n*" policies[1]. Students use spreadsheets and tree diagrams to compute and understand expected turn scores for such strategies.

Hog (a.k.a. *Fast Pig* [8]) is a variant of pig where one has a single roll each turn, but may roll any number of dice at once. This is equivalent to a game of Pig where one must commit to a number of rolls at the beginning of one's turn. Hog is described in [13], along with a 2-3 class period lesson plan and assessment guides. The core teaching ideas, analysis, and computer simulations are extended in [2] and [7].

Indeed, there are many other examples of mathematical pedagogy that make good use of Pig and its variants[2]. In the following sections, we provide examples of computer science uses of Pig, sharing exercises and online resources.

CS1 EXERCISES

Pig has a surprising number of uses in teaching fundamental programming concepts such as random number generation, nested loops, if-else chain decisions, and arrays.

[1] http://www.nzmaths.co.nz/Statistics/Probability/greedypig.aspx

[2] http://www.nzmaths.co.nz/Statistics/Probability/greedypig.aspx

The implementation of a simple, text-based Pig game can be approached in stages with each stage illustrating basic concepts of program control flow. A simple computer player could follow a "hold at 20" policy, which maximizes the expected number of points per turn, but does not maximize the expected probability of winning. Knizia describes the odds-based analysis for the "hold at 20" policy, where each roll is viewed as a bet:

> … we know that the true odds of such a bet are 1 to 5. If you ask yourself how much you should risk, you need to know how much there is to gain. A successful throw produces one of the numbers 2, 3, 4, 5, and 6. On average, you will gain four points. If you put 20 points at stake this brings the odds to 4 to 20, that is 1 to 5, and makes a fair game. … Whenever your accumulated points are less than 20, you should continue throwing, because the odds are in your favor. [9, p. 129]

This "hold at 20" policy can form the basis of a number of CS1 exercises:

1. Simulate a single turn of a computer Pig player that holds when the turn total reaches 20. Output each die roll and the final score gain for the computer player on separate lines.
2. What are the probable outcomes of the "hold at 20" policy? Run this simulation 1,000,000 times (without dice roll and final score output), counting occurrences of the possible score gains in an array. Output each possible gain on a separate line along with the estimated probability of achieving it when holding at 20.
3. It does not make sense to continue rolling when holding would win the game. Modify your simulation so that the computer player is given a non-negative `score` < 100 from standard input, and holds when the turn total reaches 20 or the goal score of 100 would be achieved. Output each dice roll, the final score gain, and the score at the end of each turn for the computer player on separate lines.
4. Using a variable `score` initialized to 0, simulate a game played to 100 points by a computer player with this policy.
5. How many turns on average does it take for the "hold at 20" player to reach 100? Run this simulation 1,000,000 times (without previous output), and output the average number of turns the computer takes to reach the goal score.
6. Modify the previous game simulation so that two computer players are competing with the same policy. At the beginning of each turn, output both scores and which player is currently playing, along with previous roll and score output.
7. There is an advantage to going first in Pig, but how big an advantage is it for "hold at 20" play? Run this simulation 1,000,000 times (without previous output), and output the fraction of games won by the first player.
8. Modify your two-computer simulation so that one of the players is randomly chosen to be the user. Prompt the user for decision from standard input, with an empty string (return/enter) indicating a decision to roll, and any other input line indicating a decision to hold.

These and other exercises allow students to see a solution to a more complex problem approached through simple incremental development. This is an especially good exercise for writing nested loops. When presenting these exercises, we first describe the goal, the final exercise, and outline the algorithm top-down (i.e. game → turn→ roll and decision), so that students can see how these algorithmic building blocks are selected and composed bottom-up.

One can experiment with many different statistics (e.g. how many rolls before a 1 is rolled) and play policies (e.g. hold after *r* rolls). These are but a sampling of a rich set of exercises which focus on control flow and simple statistics.

Piglet

Similar exercises can also applied to a simplification of Pig called Piglet[3]. Piglet is very much like Pig except it is played with a coin rather than a die. The object of Piglet is to be the first player to reach 10 points. Each turn, a player repeatedly flips a coin until either a "tail" is flipped or the player holds and scores the number of consecutive "heads" flipped. At any time during the player's turn, the player is faced with two choices: *flip* or *hold*. If the coin turns up tails, the player scores nothing and it becomes the opponent's turn. Otherwise, the player's turn continues. If the player chooses to *hold*, the number of consecutively flipped heads is added to the player's score and it becomes the opponent's turn.

As one can see, there are many introductory programming exercises that can be based on Pig and its variants.

PIG IN ARTIFICIAL INTELLIGENCE

In the following sections, we describe how Pig has been used to teach fundamental concepts of reinforcement learning[20, 18] in the context of an introductory artificial intelligence course. We outline the project, "Solving the Dice Game Pig: an introduction to dynamic programming and value iteration", which is part of a larger project *Machine Learning Laboratory Experiences for Introducing Undergraduates to Artificial Intelligence (MLExAI)*[4]. An overview of this NSF-funded work and samples of other course materials developed under this grant are published in [17, 12].

There are no prerequisite mathematics courses for this material. However, students should have a familiarity with summation notation and a programming language with syntax similar to that of Java.

Optimal Play of Pig

Although we have previously described a "hold at 20" policy for maximizing expected points per turn, this is far from optimal play. Playing to score is not the same

[3] Piglet was independently devised in [15] and in [18], where it is cleverly called Pig Tails.

[4] http://uhaweb.hartford.edu/compsci/ccli/

as playing to win. Indeed, to win more often, one must be willing to lose with a lower score, taking great risks when the opponent has a great score advantage.

At any time, when faced with the decision to roll or hold, one should *choose the action that maximizes the expected probability of winning*. Let $P_{i,j,k}$ be the player's probability of winning if the player's score is i, the opponent's score is j, and the player's turn total is k. In the case where $i + k \geq 100$, $P_{i,j,k} = 1$ because the player can simply hold and win. In the general case where $0 \leq i, j < 100$ and $k < 100 - i$, the probability of an optimal player winning is

$$P_{i,j,k} = \max(P_{i,j,k,roll}, P_{i,j,k,hold})$$

where $P_{i,j,k,roll}$ and $P_{i,j,k,hold}$ are the probabilities of winning if one rolls and holds, respectively. These probabilities are given by:

$$P_{i,j,k,roll} \quad = \quad \frac{1}{6}\left((1 - P_{j,i,0}) + \sum_{r=2}^{6} P_{i,j,k+r}\right)$$

$$P_{i,j,k,hold} \quad = \quad 1 \; - \; P_{j,i+k,0}$$

The probability of winning after rolling a 1 or holding is the probability that the other player will not win beginning with the next turn. All other outcomes are positive and dependent on the probabilities of winning with higher turn totals.

Dynamic Programming

In their book, *Reinforcement Learning: an introduction* [20], Sutton and Barto describe three fundamental classes of algorithms for reinforcement learning: dynamic programming (DP), Monte Carlo methods, and temporal-difference (TD) learning. TD learning can be viewed as a hybrid of DP and Monte Carlo methods, so DP methods are a natural starting point for teaching reinforcement learning.

Sutton and Barto define dynamic programming (DP) as a collection of algorithms for solving Markov decision processes (MDPs) [20, p. 89], including the value iteration algorithm. However, in computer science, the term "dynamic programming" commonly refers more generally beyond MDPs to a common programming pattern where one "[solves] an optimization problem by caching subproblem solutions (memoization) rather than recomputing them."[16][5]

This important general programming pattern is usually taught in a data structures and algorithms course, but we use it as a bridge for teaching value iteration. In [5], the development of a dynamic programming algorithm is described as having four steps:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

This is, in short, a recursive solution with caching and reuse of subproblem solutions.

[5] One can view value iteration as a type of dynamic programming if the cached "subproblem solutions" are state values indexed by iteration, and the number of iterations is fixed.

A recursive solution structure assumes that subproblem dependencies are acyclic. However, equations for optimal play of Pig are cyclic. Suppose two players roll a 1 in succession. Then nothing has changed about the state of the game; players have cycled back to the same situation. Such cyclic possibilities in play mean that we cannot solve optimal play of Pig using recursion.

However, a slight modification to the rules of Pig, described in [3], yields an acyclic set of equations for optimal play. We call this variation Progressive Pig, because one progresses towards the score goal each turn. Progressive Pig is identical to Pig except that a player always scores at least 1 point each turn:

- If the player rolls a 1, the player scores 1 point and it becomes the opponent's turn.
- If the player rolls a number other than 1, the number is added to the player's turn total and the player's turn continues.
- If the player holds, the greater of 1 and the turn total is added to the player's score and it becomes the opponent's turn.

Thus the equations for the probability of winning Progressive Pig with optimal play are:

$$P'_{i,j,k} = \max(P'_{i,j,k,roll}, P'_{i,j,k,hold})$$

$$P'_{i,j,k,roll} = \frac{1}{6}\left((1 - P'_{j,i+1,0}) + \sum_{r=2}^{6} P'_{i,j,k+r}\right)$$

$$P'_{i,j,k,hold} = 1 - P'_{j,i+\max(k,1),0}$$

In our curricular materials, we first illustrate the DP programming pattern in Java with the computation of Fibonacci numbers. Then we further illustrate DP with the computation of optimal play for Progressive Pig, as an approximation to optimal Pig play. We then challenge students to apply DP to the computation of optimal play for the following Pig-related exercises:

Pig Solitaire: To win, reach a given goal score g within *n* turns.

Pig Solitaire 2: Maximize one's score in *n* turns.[6]

THINK Solitaire: Compute a policy to maximize expected score in the game of THINK. THINK is identical to Pig, except that

- Two standard dice are rolled. If neither shows a 1, their sum is added to the turn total.
- If a single 1 is rolled, the player's turn ends with the loss of the turn total.
- If two 1's are rolled, the player's turn ends with the loss of the turn total *and score*.
- Each player gets only five turns, one for each letter of THINK.
- The highest score at the end of five turns wins.

[6] This is confirmation of the "hold at 20" policy in disguise.

Value Iteration

The value iteration algorithm iteratively improves estimates of the value of being in each state. In describing value iteration, we follow [20], which we also recommend for further reading. We assume that the world consists of *states*, *actions*, and *rewards*. The goal is to compute which action to take in each state so as to maximize future rewards. At any time, we are in a known state *s* of a finite set of states $\mathcal{S}$. There is a finite set of actions $\mathcal{A}$ that can be taken in any state. For any two states $s, s' \in \mathcal{S}$ and any action $a \in \mathcal{A}$, there is a probability $\mathcal{P}^a_{s,s'}$ (possibly zero) that taking action *a* will cause a transition to state s'. For each such transition, there is an expected immediate *reward* $\mathcal{R}^a_{s,s'}$.

We are not just interested in the immediate rewards; we are also interested to some extent in future rewards. More specifically, the *value* of an action's result is the sum of the immediate reward plus some fraction of the future reward. The *discount factor* $0 \leq \gamma \leq 1$determines how much we care about expected future reward when selecting an action.

Let *V(s)* denote the estimated value of being in state *s*, based on the expected immediate rewards of actions *and* the estimated values of being in subsequent states. The estimated value of an action *a* in state *s* is given by:

$$\sum_{s'} \mathcal{P}^a_{s,s'} \left[\mathcal{R}^a_{s,s'} + \gamma V(s') \right]$$

Since any action can be chosen, the optimal choice is the action that maximizes this estimated value:

$$\max_a \sum_{s'} \mathcal{P}^a_{s,s'} \left[\mathcal{R}^a_{s,s'} + \gamma V(s') \right]$$

This expression serves as an estimate of the value of being in state *s*, that is, *V(s)*. In a nutshell, value iteration consists of revising the estimated values of states until they converge, i.e., until no single estimate is changed significantly. The algorithm is given as Algorithm 1.

```
For each $s \in S$, initialize $V(s)$ arbitrarily.
Repeat
    $\Delta \leftarrow 0$
    For each $s \in S$,
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{s,s'} \left[ \mathcal{R}^a_{s,s'} + \gamma V(s') \right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \varepsilon$
```

Algorithm 1: Value iteration

Algorithm 1 repeatedly updates estimates of $V(s)$ for each s. The variable Δ is used to keep track of the largest change for each iteration, and ε is a small constant. When the largest estimate change Δ is smaller than ε, we stop revising our estimates.

When $\gamma = 1$, and all rewards are 0 except for game winning state transitions with a reward of 1, then state values are identical to win probabilities. The computed policy is optimal, maximizing the expected probability of winning against perfect competition. Value iteration's convergence, however, is not guaranteed in general when $\gamma = 1$. Convergence is guaranteed only when $\gamma < 1$ and rewards are bounded [14, §13.4]. However, in the cases of Piglet and Pig, value iteration happens to converge.

In order to apply value iteration to Piglet, we reduce the 2-agent problem to a single agent problem through our previous observation that the probability of winning after the end of a turn is the probability of the opponent losing. That is, the probability of winning after the turn's end is 1 minus the probability that we would win as the other player. With this small modification to the Bellman equations, we illustrate the solution to optimal play for Piglet in Java. We then provide the following exercises:

Pig: Compute optimal play for Pig.

Pig Solitaire 3: Compute optimal play to minimize the number of turns taken to reach a given goal score *g*.

Pass the Pigs: Compute optimal play for Pass the Pigs®. Pass the Pigs® (a.k.a. Pigmania®) is a popular commercial variant of Pig which involves rolling two rubber pigs to determine the change in turn total or score[7]. Students are given roll data for estimating roll probabilities. From this, approximately optimal play can be computed.

Additional curricular materials and full details for all dynamic programming and value iteration exercises are available through the MLExAI project website[8].

Assessment

This project was first assigned to Gettysburg College students in the fall of 2004. Five students were independently interviewed and surveyed by an external evaluator for the NSF. Results were very positive. On a 1-5 scale ("strongly disagree", "disagree somewhat", "neither agree nor disagree", "agree somewhat", "strongly agree"), students responded to questions in Figure 1.

Although this sample size is very small, these preliminary results are encouraging. The curricular materials were revised according to recommendations by student participants and reviewing faculty. Additional broader assessment of the revised materials is forthcoming as the revised materials are currently being tested across a diverse set of institutions.

[7] Rules: http://www.hasbro.com/common/instruct/PassThePigs.PDF

[8] http://uhaweb.hartford.edu/compsci/ccli/

Question	**Mean**
Requirements for the student project were clearly presented and easy to follow.	4.6
The time allowed for the completion of the project was sufficient.	4.6
The student project was interesting to work on.	4.4
The student project contributed to my overall understanding of the material in the course.	4.8
The student project was at an appropriate level of difficulty given my knowledge of computer science and programming.	4.6
After taking this course I feel that I have a good understanding of the fundamental concepts in Artificial Intelligence.	4.4
Based on my experience with this course, I would like to learn more about the field of Artificial Intelligence.	4.2
The Artificial Intelligence problem solving techniques covered in this course are valuable.	4.8
I have a firm grasp of the problem solving techniques covered in this course.	4.4
I am confident that I can identify opportunities to apply these problem solving techniques.	4.2
I am confident that I can apply these problem solving techniques to different problems.	4.2
I had a positive learning experience in this course.	4.8

Figure 1: Preliminary NSF Assessment

PIG IN OTHER ADVANCED TOPICS

Scientific Visualization

The computed optimal policy for pig is useful as a visualization exercise as well. The policy can be represented as a three dimensional array with the two players' scores on two axes and the turn total on a third. Each element of the array is a single one or zero indicating if the player should roll or hold. The data set is interesting for these exercises for two reasons. It is large enough to have interesting features, but small enough to be filtered and displayed quickly.

For a visualization of this data set students are asked to generate an isosurface for the roll-hold boundary using the Marching Cubes Algorithm described by Lorensen and

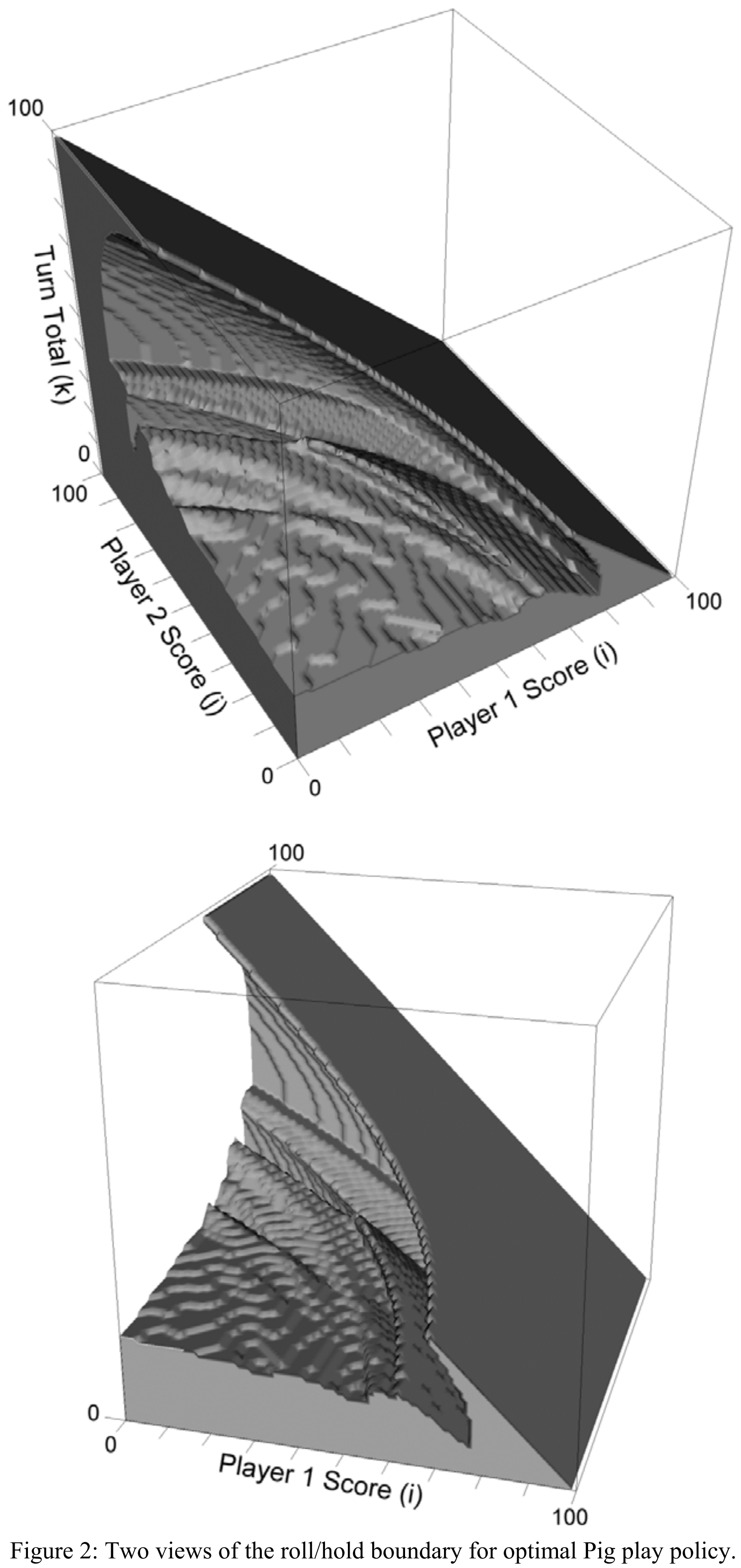

Figure 2: Two views of the roll/hold boundary for optimal Pig play policy.

Cline [11]. The resulting surface is shown in Figure 2. This surface can then be processed by smoothing or decimating the surface. Students can then compare the initial surface with the processed one. The surface for the Pig solution is such that the students should note a difference, not just in appearance of the result, but also in render times.

Networking

As a simple multiplayer game Pig also lends itself well to projects in computer network courses. Since the game is relatively easy to implement, the students can focus on building the networking portions of the project. The most obvious version requires the student to create an application in which two peers will play the game against each other.

More complex versions can be built using versions of the game that involve more than two players.

PIG IN UNDERGRADUATE RESEARCH

For simplicity, we have reduced a 2-agent problem to a single-agent problem for solving Pig. In fact, there is no need to do so. Pig is what is called a *stochastic* or *Markov game* [10], and multi-agent reinforcement learning for such games is an active area of current research. Markov games allow simultaneous player actions, so Pig is a simple type of Markov game where one player has actions {*roll, hold*} and all other player(s) have only a single *null action*. At the time of writing, optimal play for 3-player Pig is yet unsolved. Computing such optimal play in a Markov game framework would be an interesting undergraduate research project.

Although it would not be optimal, suppose we use our 2-player optimal policy to play *n*-player Pig, by treating all opponents as a single abstracted opponent with a score defined as the maximum score of all opponents. How does 3-player optimal play compare to such abstracted 2-player play? There are many such questions to explore.

Also, being able to compute optimal play for two-player Pig/Piglet allows for interesting comparisons between different reinforcement learning (RL) methods. For example, one advantage of Monte Carlo and temporal-difference learning techniques is that they rely on simulation and do not require a complete model. However, seldom visited game states are slow to converge to their correct values. In an advanced undergraduate course on machine learning, Pig and Piglet were effectively used to compare and contrast the characteristics of such reinforcement learning techniques. Generalization of single-agent RL techniques to multi-agent RL techniques provided more good undergraduate research opportunities.

As the number of players increases, the state space grows exponentially, and memory becomes an issue. In such cases, there is a natural motivation to explore function approximators (e.g. neural networks) in the context of reinforcement learning.

As one can see, a little curiosity and a simply-defined, rich problem such as Pig provide many possible good explorations for undergraduate researchers.

CONCLUSION

Simple examples are teaching treasures. In this paper, we have shown how the simple jeopardy dice game of Pig has been used in many educational settings, from middle school mathematics to introductory computer science to advanced CS courses and undergraduate research. This is the first computer science collection of Pig exercises and online resources.

Pig problems provide plenty of pedagogical possibilities and potentials perhaps not previously pondered. We hope the reader will keep Pig in mind, and perhaps find other interesting pedagogical uses for Pig.

ACKNOWLEDGEMENT

This work is supported in part by National Science Foundation grant *DUE CCLI-A&I Award Number 0409497*.

REFERENCES

[1] Robert Charles Bell. *Board and Table Games from Many Civilizations*. Dover Publications, Inc., New York, New York, USA, revised edition, 1979.

[2] James F. Bohan and John L. Shultz. Revisiting and extending the hog game. *The Mathematics Teacher*, 89(9):728–733, 1996.

[3] Justin A. Boyan. *Learning Evaluation Functions for Global Optimization*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, August 1998. Available as Carnegie Mellon Tech. Report CMU-CS-98-152.

[4] Dan Brutlag. Choice and chance in life: The game of “skunk”. *Mathematics Teaching in the Middle School*, 1(1):28–33, April 1994. ISSN 1072-0839.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, USA, 2 edition, 2001.

[6] Diagram Visual Information Ltd. *The Official World Encyclopedia of Sports and Games*. Paddington Press, London, 1979.

[7] Larry Feldman and Fred Morgan. The pedagogy and probability of the dice game hog. *Journal of Statistics Education*, 11(2), 2003. www.amstat.org/publications/jse/v11n2/feldman.html

[8] Dan Fendel, Diane Resek, Lynne Alper, and Sherry Fraser. *The Game of Pig. Teacher's Guide. Interactive Mathematics Program, year 1*. Key Curriculum Press, Berkeley, California, USA, 1997.

[9] Reiner Knizia. *Dice Games Properly Explained*. Elliot Right-Way Books, Brighton Road, Lower Kingswood, Tadworth, Surrey, KT20 6TD U.K., 1999.

[10] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conf. on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994.

[11] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(3):163–169, July 1987.

[12] Zdravko Markov, Ingrid Russell, Todd Neller, and Susan Coleman. Enhancing undergraduate AI courses through machine learning projects. In *Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference (FIE'05)*, Indianapolis, IN. IEEE Press, October 2005.

[13] Mathematical Sciences Education Board. Measuring up: Prototypes for mathematical assessment, 1993.

[14] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, New York, USA, 1997.

[15] Todd W. Neller and Clifton G.M. Presser. Optimal play of the dice game pig. *The UMAP Journal*, 25(1):25–47, 2004.

[16] NIST. Dictionary of algorithms and data structures, 2005. http://www.nist.gov/dads/

[17] Ingrid Russell, Zdravko Markov, Todd Neller, Michael Georgiopoulos, and Susan Coleman. Unifying an introduction to artificial intelligence course through machine learning laboratory experiences. In *Proceedings of the 25th American Society for Engineering Education Annual Conference and Exposition (ASEE'05)*. ASEE Press, June 2005.

[18] Stuart Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, Upper Saddle River, NJ, USA, 2 edition, 2003.

[19] John Scarne. *Scarne on Dice*. Military Service Publishing Co., Harrisburg, Pennsylvania, USA, April 1945. Second, Revised Printing.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an introduction*. MIT Press, Cambridge, Massachusetts, 1998.