# Nannon™: A Nano Backgammon for Machine Learning Research†

**Jordan B. Pollack**
Computer Science Department
Brandeis University
Waltham, MA 02454
pollack@cs.brandeis.edu
http://demo.cs.brandeis.edu

**Abstract- A newly designed game is introduced, which feels like Backgammon, but has a simplified rule set. Unlike earlier attempts at simplifying the game, Nannon maintains enough features and dynamics of the game to be a good model for studying why certain machine learning systems worked so well on Backgammon. As a model, it should illuminate the relationship between different methods of learning, both symbolic and numeric, including techniques such as inductive inference, neural networks, genetic programming, co-evolutionary learning, and reinforcement learning based on value function approximation. It is also fun to play.**

## 1 Introduction

Backgammon is an ancient game which is still popular in many parts of the world. Although it is based on a lucky device - the roll of dice to limit each player's moves - humans have discovered a wide range of strategies and skills, filling up many books with acquired backgammon knowledge, both folk and mathematical (Jacoby 1970, Magriel 1976). Its popularity soared in the US with clubs and pub tournaments in the late 70's, and it is growing in popularity again, online.

Backgammon has also become an object of study for computational gaming, as a stochastic rather than deterministic game like chess. However, the difficulty of coding all the arcane rules, especially regarding forced moves and bearing off - makes computer logic for the game run to several pages of impenetrable logic which is difficult to fully debug. Also, the breadth of the game tree prohibits deep look ahead, because rolling doubles, which allow 4 checkers to move, causes combinatorial explosion.

Nevertheless by the mid seventies it was possible to write backgammon programs on that era's IBM 360 computers. Such a player could make reasonably proficient moves. It comprised a legal move generator, a set of measurement and position testing functions, and parameter based methods to rank positions based on rough heuristics for determining game phase.

One of the earliest published computer players was built by Hans Berliner (1977). His player was similarly based on a set of hand-built polynomials over measurements of positions, as well as logical functions to determine which "phase" of a game the player was in; However, Berliner went further to include smoothing mechanisms after noticing that the computer player could be exploited as it wavered between strategic boundaries. With further work, his BKG became a respectable computer player for humans to train against.

Backgammon next became a domain for scaling up neural network learning, e.g. back Propagation (Rumelhart Hinton & Williams, 1986). Gerald Tesauro wrote a series of influential papers on training back-propagation networks to become value estimators for backgammon positions. A player can be made by combining a value estimator with a greedy algorithm which looks at all possible moves for a given dice roll, and picks the highest scoring position for the current player. His early Neurogammon approach used encyclopedic tables drawn from human tournaments. Later, it was extended with contrast-enhancing techniques (Tesauro 1987, 1989). Then, in 1992, using large scale computing power provided by IBM Yorktown Heights, he published a breakthrough paper on learning backgammon via self-play using the method of temporal differences. (Sutton 1989, Tesauro 1992). After manually increasing the set of primitive features, and using multi-ply search, TD-gammon was recognized as one of the top players in the world. (Tesauro 1995). The success of TD-gammon stimulated a lot of research in Reinforcement Learning for the rest of the decade, as well as drove acceptance of commercial programs providing analysis and challenge for professional gambling and tournament play, such as Jellyfish and Snowie.

Our work on co-evolutionary algorithms began in the early 90's (Angeline & Pollack 1993) as part of a search for clear evidence that software could be a medium for the kind of open-ended evolution of complexity seen in the

† Nannon is a copyrighted game, but may be used for research and academic purposes. Nannon is a trademark of Nannon Technology corp., which provided permission to publish the rules and board in this paper.

"arms-race" phenomena in Nature. In co-evolution, learners face a dynamically changing environment, usually composed of other learners, such that as some improve, the challenges for others would automatically increase. Besides Hillis's work on Sorting networks, Axelrod's IPD GA experiment, and Ray's Tierra model, we considered Tesauro's TD-Gammon to be indicative of successful Co-evolution, since it improved by essentially increasing the difficulty of the learning environment as it progressed. However, the fact that it used a population of 1 caused some cognitive dissonance because most co-evolutionary systems based on Genetic Algorithms or Genetic Programming used populations in the 100's.

In 1998, Alan Blair and I did a small experiment based on a validated backgammon legal move generator provided by Mark Land. We used 1+1 hill-climbing on a neural network as a value estimator. Using the current network as Champion, we added random noise to the weights and had it compete against the champion. Despite the simplicity of this algorithm, we substantially replicated the co-evolutionary learning effect of TD-gammon, although our player was not as good as the ones derived by Tesauro.

In that and subsequent work, we started asking the question: what is it about backgammon, which makes complex learning possible? Learning in the backgammon domain has far exceeded success in other games which seem much easier to learn, such as TicTacToe and Othello. The Backgammon success has not been replicated in harder games like Chess and Go, although Fogel (2002) reports intriguing results in checkers.

One approach is to try to change other tasks to be more like backgammon in order to achieve better learning, for example, adding randomness to chess. Another approach is to find a simpler problem to study. A new kind of very simple game, called the Numbers game, has been valuable in illustrating co-evolutionary dynamics (Watson & Pollack, 2001; DeJong & Pollack 2002). However, the numbers game doesn't lead to the acquisition of any knowledge or strategy.

What we realized would be needed is a simpler version of Backgammon. Tesauro started the work in TD-Gammon by simply learning to bear off from an end game position. However, learning this subgame doesn't transfer much knowledge to the full game. There are other hopeful variants of Backgammon, such as Trouble, where children race in the same direction using 4 pieces but no blocking, and Hypergammon, using 3 pieces but the full rule set, however these simplifications of the game basically turn into luck-driven races with little strategic content or the volatility we think of as *turnaround dynamics*.

Backgammon, besides the balance between luck and skill, is different from games with random elements like Monopoly or Risk, which early advantages lead to winner-take-all. In Backgammon, specific dice rolls can quickly turn a game from favoring one player to the other. It is also "mixed motive" in that Humans develop symbolic strategies involving recognizing whether to play offensively or defensively, balancing competing goals to block, contain, hit, and run.

Our hope for a small game would be one which maintains all the elements of Backgammon including:

- A random element
- Turnabout Dynamics
- Occasional forfeited and forced moves
- No Draw or Stalemate possible
- Complex strategy with mixed motives
- No first player advantage.

Such a game should have an easy-to-write legal move generator, should allow researchers to compare various machine learning techniques, should allow the development of some notions of optimal play against which to measure success.[1] A simpler game should require less computer resources for study, broadening the number of researchers involved, leading to a deeper understanding of why certain kinds of learning work. In particular, we are interested in the relationship between co-evolution, reinforcement, and dynamic programming, as well as the historic division between knowledge-based symbolic learning and numeric-based control of behavior.

## 2 Introducing Nannon

Nannon is a new game that was invented to meet these goals. Its rules and conditions were chosen to minimize complexity, maximize strategic choice, maintain volatility, and remove any first player advantage. First, consider using only one random number (instead of two), providing only 2, 3, or 4 checkers (instead of 15) per player and using a board from 4 to 12 spaces long (instead of 24).

Because of the availability of 6-sided dice, I settled on a 6-point board, with 3 checkers per side, although the game admits a whole family of games of related sizes and different dynamics. Like backgammon, players move in opposite directions, with a goal of getting all checkers off the board and out of play, while hitting their opponents back to the beginning to start over.

The game starts in an initial position, then each player takes a turn by rolling a die and if possible, moving one of their checkers the number of steps shown on the die, or off

[1] Hypergammon admits a 200 Megabyte table of positions calculated by GNUBG which allows for value function to be approximated in several days of CPU time.

the board to safety. The initial position was chosen to increase strategic interaction.

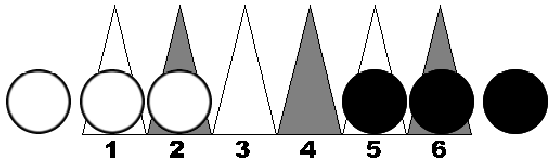



**Figure 1: The initial position where White is moving right, Black is moving left.**

The goal is to get all one's checkers across the board and out of play ("to safety"), but like in backgmammon, intermediate goals are hitting and blocking your opponent, overcoming the luck of the die with strategic choices. Consider if black rolls a 2, and moves the piece from the 5 to the 3 position:

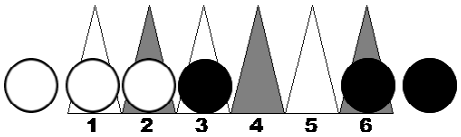



**Figure 2: Black rolls a 2 and makes a bad move, exposing two men instead of preserving a prime.**

Hitting means landing a checker on an opponent checker and sending it back to the beginning ("home").[2] If white rolls a 3, the player can move onto the board, hitting back the black piece (to the "7" position).

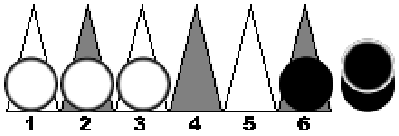



**Figure 3: White rolled a 3 and hit Black back to the 7.**

With only 3 checkers, our core realization was that since a "point" in Backgammon requires two checkers on a space, and blocking requires several or even 6 points in a row, any reduced checker game with the full rules cannot maintain blocking, which is a core strategic element of Backgammon.

So how can blocking be brought back into the reduced game? The answer we arrived at is to use *adjacency* to create a block. If a player can locate two or three checkers next to each other, we declare the other player cannot land on or hit those checkers. Therefore, the 3 white checkers above protect each other from getting hit, and block black from moving on certain rolls.

Three checkers blocking one checker would cause a forfeited turn only 50% of the time[3]. In the position of Figure 3, Black would forfeit 33% of the time, upon rolling a 4 or 5. (Black's checker on the 6 can move to safety with a 6.)

We made a second important rule decision which simplified the board representation. We decided that only one checker ever allowed on a space - e.g. no stacking of checkers at all! But what happens if the dice would allow one checker to land on another? There are 3 alternatives rules: It cannot move, it skips forward (which accelerates the game), or you are forced to hit yourself (which is quite odd!). We chose the simplest idea; a checker cannot land on another checker of the same color. Thus, in the current position of Figure 3, White rolling a 2 cannot stack the checker from the 1 to the 3 position, but must move from the 2 or 3 point.

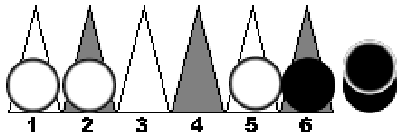



**Figure 4: White moved 2. If Black rolls a two and hits the White checker on the 5-point, the game cycles back to the initial position.**

This no-stacking rule simultaneously increased the effectiveness and importance of blocking, created forced bad rolls which break up blocks, and made the legal move generator extremely simple, as seen in the Matlab and Java examples given in the Appendix.

To find legal moves for a player, we first compute which of the 6 board positions are blocked by either a player's own checkers, or by opponent checkers which are adjacent. Then we simply calculate which of the player's 3 checkers still in play can land on a non-blocked space or escape off the board.

Of course, developing over thousands of years, Backgammon has many rules which control the emergent issues that arise during the game. For example, you need to have all the pieces off the bar in order to move any other piece, you need have all checkers in the Home quadrant before bearing off, and you have to move the highest number if you have a choice of forced moves between two dice. These rules are unnecessary or lead to stalemate in Nannon.

### 2.1 Starting Position and First Player Advantage

We represent a position as two sorted triples, of the locations of each player's checkers. The board positions are 1-6, and we use 0 to represent player 1's home and

[2] In backgammon home would be called "the bar", and no other pieces can move when any piece is on the bar. This rule doesn't make sense for Nannon.

[3] However, if we considered a rule to make a 3-point prime completely block the other player, we would end up with stalemates, which are undesirable

player 2's goal, and 7 to represent player 2's home and player 1's safety. Switching viewpoints consists of reversing the vector and subtracting it from 7. An alternative computer representation is to represent each player as a bit string using 6 bits for the location of the checkers on the board, and two or three bits to count the number of checkers which are off the bar.

We found that the default home position [000 777] was not satisfactory as it gave an overwhelming (60%) advantage for the first mover, and many games with no strategic interaction.

So the final issue in designing the game was reducing this first player advantage and increasing interaction. We looked at a variety of opening positions and rules to balance the game. We found that a starting position of [012 567] increased interaction.

## 3 Analysis of the game

Using both random play and the expert play after value function approximation, we now show that the goals for a reduced backgammon like game are satisfied.

### 3.1 Size of the game

The number of possible board states is given by the following equation, where n is the number of spaces on the board, and k is the number of checkers per player:

$$\sum_{i=0}^{k}\sum_{j=0}^{k}\binom{n}{i}\binom{n-i}{j}(k+1-i)(k+1-j)$$

Consider placing i=3 checkers of player 1, and j=1 checker of player 2 on a 6 point board, leaving 2 of player 2 checkers off the board. There are $\binom{6}{3}$ ways of placing the first 3 checkers, $\binom{6-3}{1}$ ways to place the 1 checker of the second player, and 3 ways to allocate the two remaining player 2 checkers to either home or safety.

For the 6-position, 3-checker game, this works out to 2530 states, although in practice the state where both players have 3 checkers to safety cannot be reached.

By comparison, using 3 checkers on an 8, 10 or 12-point game have 9784, 31426, and 86148 states respectively. Using 6 checkers each on a 12-point board creates a rare stalemate possibility within its 4,203,123 states. Nannon is really a parameterized set of backgammon like games.

### 3.2 No First Player advantage

Even though the raw starting position has 57% equity for player one, rolling a 4 sided die (no 5 or 6 on opening) drops the advantage to 53%. Subsequently we found a new initial roll: Both players roll their dice, and the winner gets a first roll based on the difference between the dice (e.g. 6-4=2). This lowers the retries from 1/3rd to 1/6th of the time and is fair to both players. The initial roll is biased in that 1/3rd of the time it results in a "bad" 1, and 1/15th of the time it gets a "good" 5.

Although it is a short game, and each dice roll is meaningful, the initial position and dice roll makes it so it so that the first player has no significant advantage, at 51.5%.

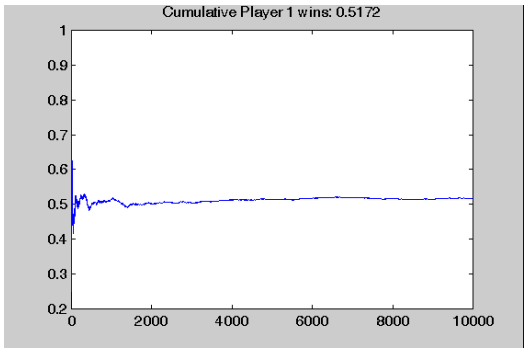



**Figure 5: In 10,000 games between optimized players, Player 1 wins about 51% of the time.**

### 3.3 Turnabout Dynamics maintained

One of the critical issues in reduced backgammon games is the loss of the volatility, or turnabout dynamics; this unpredictability about which player is going to win is essential to the popularity of the game, as it is to sports like Basketball and Soccer. Nannon allows games to reverse almost until the final few rolls. This can be seen in the following analysis of 10,000 games. We calculated the equity of player 1 at every move and count the times per game the first player equity crosses zero. Only 20% of the time does an initial lead carry through.

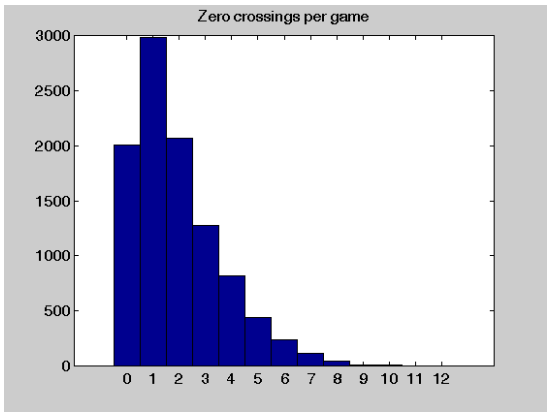



**Figure 6: Volatility is shown by the number of times the expected winner changes across a game. Calculated in 10,000 games with optimized players. X-axis is the number of flip-flops per game; Y-axis is the number of games out of 10,000.**

### 3.4 Length of game

Nannon is a fast game, with a mean of 13 rolls to completion, although long games up to 38 rolls have been observed. This enables 10's of games per second to be evaluated in a high level language like Matlab or Lisp, and 1000's in a compiled language like C or Java. Figure 7 shows the length of games out of 10,000.

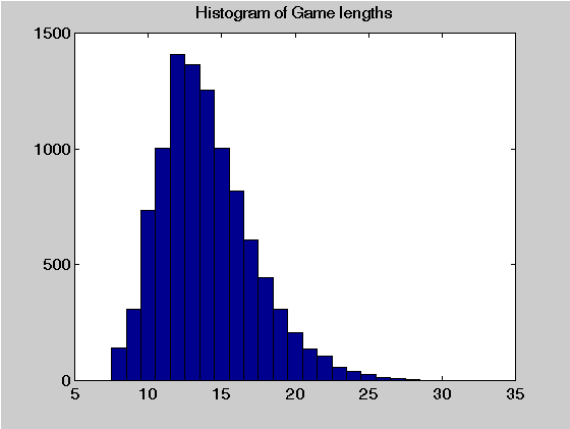



**Figure 7: Histogram of game length.**

### 3.5 Balance between Luck and Skill

Over a number of games, we calculate how many times each player forfeits a move, is forced by the die to make a specific move, or has 2 or 3-way choice. Just under 50% of the moves involve choice, as shown in the pie chart below.

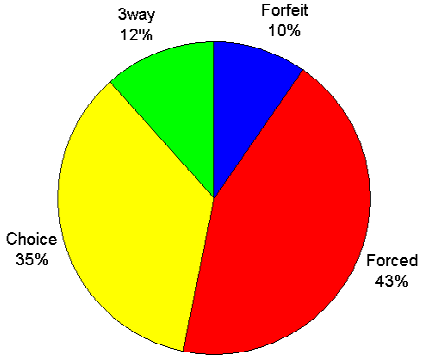



**Figure 8: Almost 50% of the time, players have a strategic choice between two and three checkers. Forfeited rolls occur when an opponent has adjacent checkers (a prime). Forced moves occur mostly when a player has only one checker left.**

### 3.6 Learnable using value function approximation

The game falls under the Bellman (1957) equation, which means there is theoretically an optimal sequential control policy based on a converged expected value for each state. The value of any state is the utility (or equity in backgammon terms) based on fair dice and future optimal play by both players. Each position can be assigned a value, and a strategy for play is simply the greedy algorithm, which looks at all moves enabled by the roll of the die and chooses the one with maximum likely reward for the current player. This is the same way that a neural network value estimator like TD-Gammon is turned into a player. Learning the symbolic rules for a game remains a hard problem.

Calculating the value function is given for ending positions – E.g. 0 or loss and 1 for win is trivial[4]. For position in a racing game, after no more contact or hitting is possible, calculating the value functions is a simple recursive application of dynamic programming. However there is a large set of positions that enable hitting to form cycles, which lead to a large system of unknowns. In many real world applications, the number of possible states is too high, but for Nannon (with a 6 point board and 3 checkers each), there are only 2530 possible positions making value function approximation eminently practical. For each state of the game, either it is an end state or we update its value by looking ahead under all dice roles for the opponent's optimal response, and multiply it by the probability of the die roll (e.g. $1/6^{th}$).

Starting with the end game positions labeled as 0 or 1, and with values exactly solved for the racing states where no further hitting is possible, in 15 passes across the 2530 states, the sum of the square of difference between values before and after each iteration rapidly dropped to $10^{-7}$.

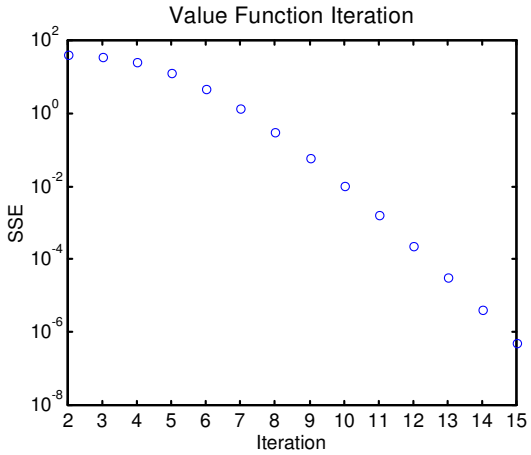



**Figure 9: Convergence of VFA in Nannon leads to an optimized player.**

[4] In actual play, the use of doubling, "gammons", and tournament rules, complicates the value calculation.

## 4 Conclusions

Although we have not yet done a wide range of machine learning experiments on the Nannon game besides value function approximation and simple heuristics based on Maslow's "Hierarchy of Needs," (like Always go to safety, then Always Hit, then Always keep block) there are many more experiments and comparisons which can be done across learning methods using this game as a model.

For example, the game can be subject to genetic programming, co-evolutionary learning, neural networks, TD learning and other reinforcement methods related to dynamic programming, as well as symbolic techniques such as Inductive inference or Inductive Logic Programming.

Backgammon, in this simpler form of Nannon is a perfectly sized test problem which ultimately could shed light on the old computational intelligence issue of whether cognition is analog and numeric based on associationism and control theory, or digital and symbolic based on universal computation.

Certainly as humans play such a game, they discuss symbolic strategies regarding when to hit, when to run, when to keep a prime versus losing tempo and so on. As expertise develops, the symbolic is infused with more statistical and numeric models to aid decision-making. Yet, according to the theory of sequential choice developed by Bellman, a greedy policy based on the converged value function should be the top player in the world (assuming fair dice).

Perhaps as our understanding of consciousness has evolved to realize that the narrative is just a story our mind constructs to explain our complex behavior based on diffuse and physical complex processes of our brains (Dennett 1991), perhaps the symbolic rules of a game is also just a story we tell as our biological organs adapt to optimize utility.

Acknowledgements

Dylan Pollack and Brad Rosenberg helped play the first few games. Anthony Bucci supplied the Java legal move code. Michael Daitzman provided much moral support and user testing. Thanks especially to Michael Littman for a discussion on VFA one evening.

## 5 References

Angeline, P. J. & Pollack, J. B. (1993) Competitive environments evolve better solutions to complex problems. Fifth International Conference on Genetic Algorithms. 264-270.

Robert M. Axelrod (1987) The evolution of strategies in the iterated prisoner's dilemma. In Genetic Algorithms and Simulated Annealing, chapter 3, pages 32--41. Morgan Kaufmann.

Hans J. Berliner (1977) Experiences in Evaluation with BKG - A Program that Plays Backgammon. IJCAI 428-433

De Jong, E.D. and J.B. Pollack (2004) Ideal Evaluation in Coevolution, Evolutionary Computation, Vol. 12, Issue 2, pp. 159-192

Dennett, D.C. (1991) Consciousness Explained. Boston: Little, Brown.

Fogel, D. B (2002) BLONDIE24: Playing at the edge of AI. San Francisco: Morgan Kaufmann.

Hillis, D. (1992). Co-evolving parasites improve simulated evolution as an optimization procedure. In Alife II: Proceedings of the 2nd International Conference on Artificial Life. Addison-Wesley.

Paul Magriel. (1976) BACKGAMMON, New York: Times Books

Oswald Jacoby & John R. Crawford. (1970) The Backgammon Book. New York: Viking.

Pollack. J B. & Blair A. (1998). Co-Evolution in the Successful Learning of Backgammon Strategy. Machine Learning, 32, 225-240.

Ray, T. S. (1991), An approach to the synthesis of life. In : Langton, C., C. Taylor, J. D. Farmer, & S. Rasmussen [eds], Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity, vol. XI, 371-408. Redwood City, CA: Addison-Wesley.

Rumelhart, DE, Hinton, GE, and Williams, RJ (1986) Learning representations by back-propagating errors. Nature, 323, 533-536.

Sutton, R.S. (1988) Learning to predict by the methods of temporal differences. Mach. Learning 3, 9-44.

Tesauro, Gerald (1992), Practical Issues in Temporal Difference Learning, Machine Learning 8, 257-277.

Tesauro, Gerald (1995) Temporal Difference Learning and TD-Gammon, Communications of the ACM, March 1995, 38(3):58-68.

Tesauro (1989): "Connectionist learning of expert preferences by comparison training", Advances in NIPS 1, 99-106.

Backgammon Varieties (2004) http://www.bkgm.com/rgb/rgb.cgi?view+846

Watson RA & Pollack JB, (2001), "Coevolutionary Dynamics in a Minimal Substrate", in GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference. Spector, L, et al, editors. Morgan Kaufmann, 2001.

## Appendices

### 5.1 Legal move generator in MATLAB

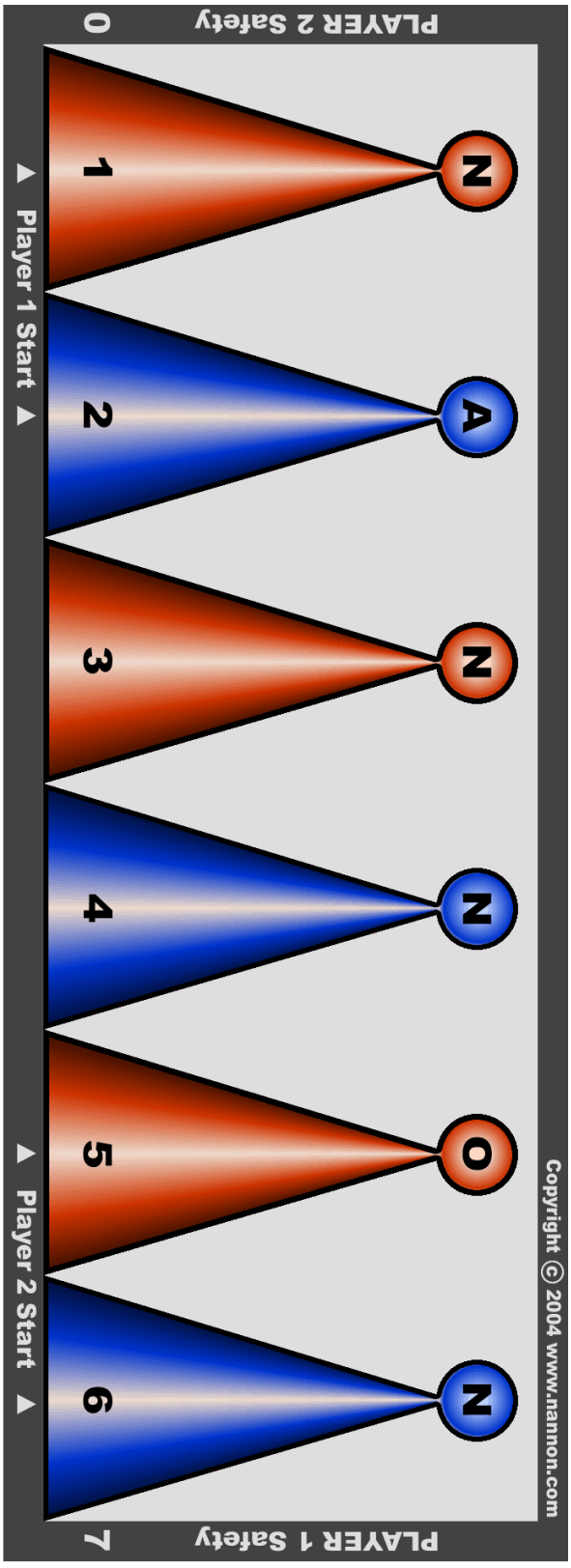
```
function moveable=legmove(pos,die)
% pos is a sixtuple [p1 p1 p1 p2 p2 p2]
% each from 0 to 7, each triple sorted
% output is a bitvector for moving
pos(1:3)
% assumes player 1 to move

moveable=zeros(1,3);
blocked=zeros(1,7);%blocked(7) is
always 0

%block adjacent opponents
%remember that pos(4 5 6) are sorted
if pos(4)<6 & pos(4)>0 &
pos(4)+1==pos(5) blocked(pos([4
5]))=1;end;
if pos(5)<6 & pos(5)>0 &
pos(5)+1==pos(6) blocked(pos([5
6]))=1;end;

%block my own checkers on the board
for i=1:3
   if mod(pos(i),7)
blocked(pos(i))=1;end
end
%Calculate unique unblocked moves
for j=1:3
   if pos(j) ~= 7 % once in safety
don't move
      if j==3 | pos(j) ~= pos(j+1)
%stop duplicate 0 choices here
         if
~(blocked(min(7,pos(j)+die)))
moveable(j)=1;
         end
      end
   end
end
```

### 5.2 Printable Board



**5.3 Legal move generator in Java**

```
/*
   we store the board in two ints,
   m_black and m_red which look like
this
   (take careful note of the indexing;
red is indexed
   backwards w.r.t. black);

   b b b 0 B B B B B B 0
   0 1 2 3 4 5 6 7 8 9 10

   b is the home
   B is the board
   0 are for efficient legal move calc.

   we'll index this in two ways: with
pos and with idx
   (position and index, resp).  idx
indexes the bits in
   the int, so starts from 0 and runs
to NUM_PIECES + BOARD_SIZE + 2 - 1
   (2 for the pads).  pos indexes the
board, starting from 0
   and running to BOARD_SIZE - 1.
negative positions indicate the
   bar; -1 is the 0, -2 is the bar, -3
is the bar, etc.*/

public class Board {
    // handy constants
    public static final int NO_ONE = -
1;
    public static final int BLACK = 0;
    public static final int RED = 1;
    public static final int NUM_PIECES
= 3;
    public static final int BOARD_SIZE
= 6;

    // board state
    int m_black;
    int m_red;
    int m_whoseTurn;
    int m_nMoves;

    public boolean isLegal(int
nFromPos, int nDie) {
        int me = m_whoseTurn == BLACK ?
m_black : m_red;
        int opp = m_whoseTurn == BLACK
? m_red : m_black;
        int nFromIdx = nFromPos +
NUM_PIECES + 1;
        int nToIdx = nFromPos < 0 ?
nDie + NUM_PIECES : nDie + nFromIdx;
        int nToPos = nToIdx -
NUM_PIECES - 1;

        if(nFromPos < 0) {
            for(int i = NUM_PIECES-1 ;
i >= 0 ; i--) {
                if( (me & (1<<i)) != 0
) {
                    nFromIdx = i;
                    nFromPos = i -
NUM_PIECES - 1;
                    break;
                }
            }
        }

        if(nFromPos >= BOARD_SIZE)
return false;

        if( (me & (1<<nFromIdx)) == 0 )
return false;

       if(nToPos >= BOARD_SIZE) return
true;

        if( (me & (1<<nToIdx)) != 0 )
return false;

        int nOppToIdx = BOARD_SIZE +
2*NUM_PIECES + 1 - nToIdx;
        if( (opp & (1<<nOppToIdx)) != 0
&& ( (opp & (1<<(nOppToIdx+1))) != 0 ||
(opp & (1<<(nOppToIdx-1))) != 0 ) )
return false;
        }



    public int[] getLegalMoves(int
nDie) {
        Vector v = new Vector();
        for(int pos = -1 ; pos <
BOARD_SIZE ; pos++) {
            if( isLegal(pos,nDie) )
v.add(new int[]{pos});
        }

        int[] ret = new int[v.size()];
        for(int i = 0 ; i < v.size() ;
i++) {
            ret[i] =
((int[])v.elementAt(i))[0];
        }

        return ret;
    }
}
```