# An MCTS Program to Play EinStein Würfelt Nicht!

Richard J. Lorentz

Department of Computer Science,
California State University,
Northridge CA 91330-8281, USA
lorentz@csun.edu

**Abstract.** EinStein Würfelt Nicht! is a game that has elements of strategy, tactics, and chance. Reasonable evaluation functions can be found for this game and, indeed, there are some strong mini-max based programs for EinStein Würfelt Nicht! We have constructed an MCTS program to play this game. We describe its basic structure and its strengths and weaknesses with the idea of comparing it to existing mini-max based programs and comparing the MCTS version to a pure MC version.

## 1 Introduction

EinStein Würfelt Nicht! (EWN) is a fairly new game, invented in 2004 by Ingo Althöfer [1]. It is played on a 5 × 5 board where each player (called Blue and Red) starts with six pieces, numbered 1 through 6, placed in a triangular pattern as shown in Figure 1.[1] The initial numbering of the pieces is done randomly.

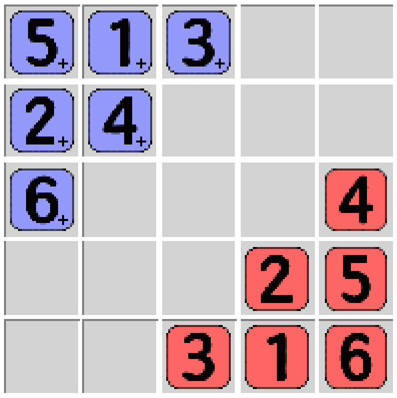

**Fig. 1.** A typical EWN starting position

Blue moves (tiles are indicated by +) first and the game is won by the first player to place one of his[2] pieces in the opponent's corner square, referred to as the goal. Pieces move one square at a time, always towards the goal, either horizontally, vertically, or diagonally. For example, this means that if a piece is not on one of the edges adjacent to his goal, it will have three legal moves. If a piece moves on top of an existing piece (his own or the opponent's) that piece is captured. The piece to move is decided by the roll of a die. In the case of Figure 1 we see that the die roll is 4 and so Blue can move his piece numbered 4 either to the south, east, or diagonally southeast. If instead a 2 had been rolled then Blue's legal moves would be either to capture his own 4 or 6

[1] All EWN figures are taken from the Little Golem web site [10].
[2] For brevity, we use 'he' and 'his' whenever 'he or she' and 'his or her' are meant.

stone or move diagonally southeast. If the die rolled corresponds to a stone that has been captured then we find the lowest numbered stone greater than the die value and the highest numbered stone less than the die value and we may move either of these stones. An alternate way to win the game is to capture all of the opponent's pieces.

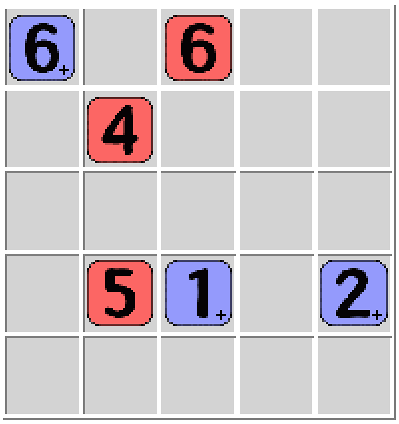

**Fig. 2.** Late in an EWN game

Consider the position shown in Figure 2. If it is Blue's turn to move, since his 3 piece is not on the board he may move either piece 2 or piece 6. There are three legal moves for piece 6, including the capture of Red's piece 4, but more interesting there is only one legal move for Blue's piece 2 and that move lands in the goal so Blue can win the game. If it is Red's turn to play he is only allowed to move piece 4, but among his three legal moves one of them is to capture Blue's 6 on the goal square, winning for Red.

EWN has both tactical and strategic elements. Near the end of the game a player must carefully calculate which pieces are likely to reach the goal square while being careful to not allow all of his pieces to become captured. But these tactical decisions require probabilistic calculations because the die determines which pieces move. Throughout the game there are a number of competing strategic ideas. Clearly having a piece near the goal increases the chances of reaching the goal. However, if the pieces numerically adjacent to the piece near the goal are still on the board then there is only a one in six chance that piece will be able to move so it is desirable to have some of your own stones captured (either by your opponent or by yourself) to increase the chance of being able to move pieces that are near the goal. This needs to be balanced with the need to prevent all of ones pieces from being captured.

This mix of strategy, tactics, and randomness, combined with the fact that games are not too long makes EWN a fun and interesting game to play.

## 2 OneStone, an EWN Playing Program Test Bed

We have written an MCTS-based program, named OneStone, to play EWN. It was created as an experiment to see how well the MCTS paradigm works with a simple game with random elements. Intuitively, this should be an ideal setting since Monte-Carlo simulations should be an effective way to deal with the randomness in the game. In fact, one of the questions we hoped to answer was whether an MCTS version will outperform a basic Monte-Carlo player. It turns out that it does, but not by very much. We were also interested in seeing how an MCTS program could compete against existing mini-max based programs. It appears to play on a par with such programs.

There are a number of existing EWN playing programs that we are aware of. Some are inactive now and most have and/or do play via the Little Golem website (LG) [8]. Fig. 3 shows a summary of these programs.

Testing ONESTONE was done via self-tests and LG. EWN has a lively presence on LG with its many human players and five other programs that play fairly regularly. However, this is a “turn based” game-playing site which means that players have considerable time to make their moves, usually 24 hours or more. This is not a particularly expeditious vehicle for testing a developing program. So, we decided to develop first a very basic version and then use that version as the basis for self-testing future versions. Finally, we took our most sophisticated version of ONESTONE and let it play on LG to see how it faired against the other programs and human players.

## 2.1 The Road to UCT

As we said above, we constructed the program incrementally. First we built a basic Monte-Carlo program that would play the game and then made a straightforward play UCB type enhancement [2] so that the more promising moves would be simulated more often. This version became our standard from which we tested all future versions. We found that even with this straightforward UCB construction ONESTONE played a reasonably strong game.

| Program name | Author | Status | Program type |
|---|---|---|---|
| FRAGGLE [9] | Ingo Schwab | inactive | mini-max with endgame tablebases |
| GAMBLER | Richard Pijl | active on LG | mini-max, in development |
| HANFRIED | Stefan Schwarz and Jörg Sameith | active on LG | mini-max |
| MEINSTEIN | Theo van der Storm | inactive[3] | mini-max |
| NAÏVE CHILD | Mark Pawlenka | active on LG | pure MC (no MCTS tree) |
| RORORO THE BOT | Phil Carmody | active on LG | mini-max, emphasis on speed |
| SYBIL | Wesley Turner | active on LG | mini-max with endgame tablebases |

**Fig. 3.** EWN playing programs

The next step was to add the tree structure for MCTS. We did this using the basic UCT algorithm [3,5,7]. But we needed to add additional structure to deal with the die throws. A typical node in a normal UCT tree will have elements corresponding to the visit counts, the win counts, and a pointer to the children. A node in the UCT tree for ONESTONE still contains a single visit count and a single win count, but also contains six separate child pointers. Conceptually, from any node we are creating six different UCT subtrees, one for each possible die throw. Since each child has an equal chance of being the selected move, during UCT expansion we make sure the children are

[3] MEINSTEIN competed in this year’s Computer Olympiad in memory of the late program’s author. It was operated by Jan Krabbenbos,

visited uniformly and then collect the visits and wins for all six children of a node into a single value.

There is a temptation to emphasize die throws corresponding to subtrees that are less well understood, e.g., that have wins/visit ratios near 50%. Similarly, as pieces start leaving the board multiple die throws will correspond to the same choice of moves and it is tempting to try to collect these into a single search. In both cases, however, there is no easy way to accomplish this while retaining the proper wins/visit ratio for the subtree's parent.

There are two natural ways to visit the children uniformly. Any time we visit a node in the tree we can randomly choose the child. In some ways this seems very much in the spirit of Monte-Carlo simulation and random die throwing. Nevertheless, we elected to do it more methodically, by keeping track in every node the last child visited and then visiting the next child when that node is visited again.

## 2.2 MCTS Improvements

There are a number of techniques used for improving MCTS performance, many of which were developed for and have been extremely successful with Go-playing programs. We implemented two of the most common ones in ONESTONE.

The first is to improve the quality of the random playouts, a suggestion that was originally articulated in [6]. The point of a random playout is to give some measure of the strength of a position, or more precisely the likelihood the player will actually win from that position. This means that the closer the playout looks like a real game, the more information it should be providing. But it is well known that improving the playouts must be done with great care. In fact, somewhat counter intuitively, reasonable looking playout "improvements" can actually make the program play worse. This is because the improved moves can introduce subtle bias that, for example, might encourage certain move sequences that are not always favorable.

We made two major modifications to our playouts. The first modification is to look for an immediately winning move (a "mate-in-one") and playing it if it is available. It does not make much sense to continue a random playout if a player is sitting on a winning move but is not playing that move. The second modification to the playouts encouraged moves that either get closer to the goal or that capture a piece. For example, in the position in Figure 2 if a 5 is the die value, it seems unlikely that moving piece 5 west is the best move. Moving north or northwest is much more likely to be the best move. For this reason, during a random playout we make it twice as likely that either of the two suggested moves is made over the move to the west. Similarly, captures, whether of an enemy piece or your own, are often interesting. Like moves towards the goal, captures are also given double the chance of being made during a random playout.

The second MCTS improvement that we made is to give "prior" initial values to new MCTS tree nodes. When creating new nodes for the MCTS tree the win counts and the total playout counts are usually initialized to zero. However, if there is prior knowledge of the quality of the move represented by a node relative to its siblings, then it sometimes makes sense to initialize these values to something other than zero that reflects the perceived relative value of that position [4]. In ONESTONE we used the same ideas here that we used for the random playouts. New MCTS tree nodes are initialized to have visit counts of 100. The win counts of moves that move towards the goal are scaled. A move that places a piece one square away from the goal is given an initial win count of 90. If two away from the goal, 75. Three away, 60. Four away gets the default value of all moves, 40, and 5 away from the goal gets a value of 20. We also continue to assume a capture is interesting, so a capture is given an initial value of 65. If a move is both a capture and a move towards the goal the initial win count is set to the maximum of the two plus 5.

## 3 EWN Variants

There are a number of variants to EWN, two of which are also played on LG. One is referred to as “Black Hole”. In this variant the square in the middle of the board is designated as the black hole in that any piece that moves to that square is immediately removed from the board. Since it is often strategically sound to capture one’s own pieces to increase the strength of the remaining pieces, this provides another means to reduce the number of one’s own pieces on the board.

The other variant is called “Backwards Capture” and it has the property that when capturing one may capture any piece adjacent to one’s own, orthogonally or diagonally, that is, in any of eight possible different directions. As will be discussed in a bit more detail below, the black hole variant appears not to change the complexity of the game very much, while backwards capture seems to make the game quite a bit different. For these experiments, our implementations for the two variants are essentially equivalent to the EWN implementation. That is, that random playout biases and the prior value settings are set the same in the variants as they are in the normal EWN game. Certainly this is not optimal for the two variants but it does give us a good feel for how the variants differ in this early stage of program development.

## 4 Results

We compared the MCTS version of the program against the basic MC version of all three variants. Each variant played 500 games as Blue, that is, moving first, and 500 games playing Red. Each player is given 30 seconds per move. The results are summarized in Table 4 where the MCTS results are shown in bold.

In our experiments we found that when playing one version against itself the blue player had a slight edge. This effect can be seen in the table where we notice that

| | MCTS playing Blue | MCTS playing Red |
|---|---|---|
| Normal EWN | **291** – 209 | 230 – **268** |
| Black Hole | **288** – 212 | 238 – **262** |
| Backwards capture | **416** – 84 | 101 – **399** |

**Fig. 4.** Results of MCTS vs. plain MC for all three variants

though the MCTS version always outperforms the MC version, when playing Blue the improvement is even greater. But the critical observation is that the MCTS versions always do outperform the basic MC version, though maybe not to the levels we had hoped.

The third line of the table we find to be quite remarkable. From the fact that the MCTS version is so much stronger than the basic MC version we may conclude that the backwards capture variant is, in some sense, a more complicated game than the other two variants and thus the MC approach is insufficient to deal with the subtleties of this variant. Here is an example of a fairly simple game where MCTS is shown to be substantially superior to Monte-Carlo.

We let our best versions of ONESTONE play on LG. On LG most of the contests are multiple game matches where the first to win 3 (or 5 or 7) games wins the match. This, in some sense, removes the first move advantage because the winner of any game in a match will move second in the next game. Since games progress so slowly on LG the data is still a bit thin, but the Tables 5 and 6 summarize the current situation. With LG games we allow ONESTONE a maximum of 5 minutes per move but in practice most moves are completed in under a minute. This will be discussed in Section 5. The opponents, of course, have more than 24 hours to make their moves but I suspect most spend about the same amount of time on their moves as ONESTONE does.

| EWN variant | wins | losses |
|---|---|---|
| Normal | 346 | 120 |
| Black Hole | 54 | 16 |
| Backwards Capture | 65 | 25 |

**Fig. 5.** OneStone results on Little Golem

| Program | wins | losses |
|---|---|---|
| GAMBLER | 14 | 14 |
| HANFRIED | 0 | 0 |
| NAÏVE CHILD | 14 | 4 |
| RORORO THE BOT | 5 | 2 |
| SYBIL | 1 | 0 |

**Fig. 6.** OneStone versus other programs

Fig. 5 shows that ONESTONE is playing a respectable game against the mostly human opponents. It has played a few games against other computer programs as shown in Fig. 6. Though there is still not a large amount of data, it appears to be playing at least even with the mini-max based programs and, not surprisingly, is doing better against the pure MC program NAÏVE CHILD.

LG provides ratings for players of its various games. Currently ONESTONE is rated 1777, placing it 15th among the approximately 400 players. Though its rating seems to average around this general value, it is interesting to note that it shows a huge degree of variation. In the past month its rating has been as low as 1625 placing it in position 100 on the overall list and it has been as high as 1880 where it was rated 3rd on the list. Such huge swings are not uncommon with EWN ratings indicating that the random element of the game is not insignificant.

## 5 Remarks

We have demonstrated that the MCTS approach to programming EWN is reasonable. Its relative success on LG shows that it is playing a quite good game. Also, the improvements provided by adding a UCT tree show that MCTS provides benefits over pure MC.

The obvious next step would be to tune the biases in the random playouts and the initial prior values in the MCTS tree. Though we do not doubt that there will be some gain in such an endeavor, early experiments seem to indicate that the improvement is likely to be minor. We find ourselves searching for other techniques. Perhaps our notion of "good moves" needs to be refined. Local properties of the piece being moved may need to be taken into account, similar to the way patterns are used in many MCTS Go-playing programs.

A second issue that needs attention is relating the wins/visits ratio with the expected win value of a node. In normal MCTS we know that the wins/visits ratio does not correlate very well with the likelihood of actually winning the game. This is not an issue with normal MCTS programs since, whatever the wins/visits ratio means, that is the value we must use when traversing the UCT tree and deciding which children to visit. In the case of EWN, things are not so clear. Certainly the best move to make corresponds to the node with the highest expected value. But will this always coincide with the node with the highest wins/visits ratio? For example, if some children of a node are reporting wins/visits ratios of 90% the reality is that those moves are almost surely wins. By "almost surely" we mean with much greater probability than 90%. More than likely, the expected value of such a node is actually greater than what is reported by the wins/visits ratio. We would like to understand this better and, ideally, adjust our move selection process accordingly.

The third issue is somewhat related to the comment above. Our experience shows that at the root of the MCTS tree if one node is visited sufficiently more often than are its siblings, there is very little chance another node will catch up any time soon. As a result we find that if we set a cut-off value for this difference and stop the search if this cut-off is reached, the search can be terminated early with no ill effects. Since OneStone is playing against humans on LG (and certainly can be made to play locally, too), early termination provides the benefit that moves can often be made quickly, a feature quite desirable to most humans when playing. Tests show that this does no harm to the playing strength of the program, yet with the proper cut-off value we now have OneStone making most of its moves in under 30 seconds, about 90% in under 1 minute, and a very few, fewer than 1%, requiring the full 5 minutes allotted. This is a useful practical feature.

The fourth issue is a sequel on the third issue. Because of the discovery mentioned in the previous paragraph we thought it might be beneficial to extend this technique to all nodes in the tree. The idea being that if the visits value of one child of a node is sufficiently ahead of its siblings we discontinue any searches of the children. Of course, this is an unsafe operation, but it may have value in practice. We experimented with various approaches, such as uniformly using the same cut-off throughout, graduating the cutoff according to the node's depth in the tree, etc. Most approaches did neither harm, nor did they seem to benefit the program.

## References

1. Althöfer, I.: On the origins of EinStein würfelt nicht!, `http://www.althofer.de/origins-of-ewn.html`
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235–256 (2002)
3. Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M(J.) (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)
4. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: ICML 2007: Proceedings of the 24th International Conference on Machine Learning, pp. 273–280. ACM, New York (2007)
5. Gelly, S., Wang, Y.: Exploration exploitation in go: UCT for Monte-Carlo Go. In: Twentieth Annual Conference on Neural Information Processing Systems (2006)
6. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, France (2006)
7. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
8. `http://www.littlegolem.net/jsp/index.jsp`
9. `http://datendissertationschwab.de/4.html`