Life of PAI: The Pathwayz AI

Building an AI that Can Beat You at Your Own Game
CS 221 Final Report

Louis Lafair, Nikhil Prabala, and Michael Tucker
lafair, nprabala, and mictuc at stanford.edu

***Abstract*— In 2006, Louis Lafair invented a board game. In 2014, Pathwayz got published. In 2017, the inventor lost to a computer at his own game. This paper explains how we built PAI, the world's best (and only) Pathwayz AI. The challenge was the large tree size of Pathwayz, between that of Chess and Go. Building off game-playing AI literature, we explored various reflex and search models. Through bracket play of the various algorithms, we discovered which features and models worked best. Our final PAI consists of 440 smart features, with weights trained in over ten thousand games through temporal difference learning, applied in a narrowing beam minimax algorithm. We conclude by discussing results, possible sources of error, whether or not there's a first player advantage, PAI's novel approach to Pathwayz, and future work.**

I. INTRODUCTION

Pathwayz is a 2-player strategy board game, invented by Louis Lafair and published in 2014. The popular game has been sold online, at Barnes & Noble, and in hundreds of stores across North America. Pathwayz won Dr. Toy Best Pick of 2014 and was recommended by Mensa in 2015.

The goal of Pathwayz is to build a path across the long side of an 8x12 board. Each turn, you can place a regular piece in your color, or a permanent piece in your opponent's color to flip the surrounding spaces. The full instructions are here, and a sample game is here.

We built a GUI for efficient and informative testing. Our GUI is hosted here you can play Pathwayz against other humans and AIs.

In the spirit of AlphaGo, we wanted to make the world's best PAI (Pathwayz AI). Our main challenge was game complexity. Pathwayz is reminiscent of Chess and Go. Its tree size is larger than that of Chess. Its tree breadth is halfway between that of Chess and Go.

In order to handle such a complex game tree, we experimented with different approaches: reflex models (Random, Baseline, Advanced Baseline, Dumb Features, Smart Features), general search models (Monte Carlo Search, Monte Carlo Tree Search), and adversarial search models (Expectimax, Beam Expectimax, Minimax, Beam Minimax, and Narrowing Beam Minimax). We used temporal difference learning pipelines to train feature weights. Below, we provide details about how we implemented each of these algorithms.

Fig. 1: Pathwayz GUI (white player just won this game)

Game	Breadth	Depth	Size ($\text{breadth}^{\text{depth}}$)
Pathwayz	135	60	6.60×10^{127}
Chess	35	80	3.35×10^{123}
Go	250	150	4.91×10^{359}

Fig. 2: Pathwayz complexity compared to Chess, Go

II. PRIOR WORK

There exists a substantive body of work around game-playing AI for similar games. Approaches in the field span from standard tree search to deep reinforcement learning. Different approaches have been employed to create strong AIs for games including Go,

Chess, Othello, and Backgammon. Most notable among these is the Go player AlphaGo, whose defeat of Grandmaster Lee Sedol irrevocably altered the landscape of game-playing AI with its combination of Monte Carlo Tree Search and deep neural networks [3]. AlphaGo's success resulted from this paradigm: simulating full games yields better results than searching a game tree with preset knowledge.

Granted, more traditional search and explore algorithms, based on preset knowledge, can also perform well. The Chess AI DeepBlue obtained resounding success against Gary Kasparov in 1997 only by searching through a database of previously seen grandmaster games [2]. DeepBlue's approach relied on weighting the combined wisdom of centuries of Chess tradition, without playing any games itself. DeepBlue also employed customized hardware to cut down its search time [4]. nately, Pathwayz does not have a comparable history of games to draw upon, rendering this kind of search strategy untenable.

The younger game of Othello is a closer analog to Pathwayz in terms of existing research and literature. When Othello AI's were first developed, there was no large database of grandmaster games to draw upon only general play strategies [8]. Iago, the current Othello champion, combined domain specific knowledge with a minimax search algorithm to achieve high performance. Similarly, our many iterations of PAI relied on specially chosen domain features in the absence of established strategies or long game histories.

However, an arbitrarily defined strategy lacks flexibility and performs best with an algorithm that searches at significant tree depths. Pathwayz is too complex of a game to allow for deep search in reasonable time.

Getting around this issue, TD Gammon the backgammon AI employed Temporal Difference Learning to move from a pure beginner state to competent player by playing many sample games [9]. While TD Gammon employed a neural net structure to achieve its gains, we wondered whether TD-Learning applied directly to domain specific feature weights might also provide significant performance improvements.

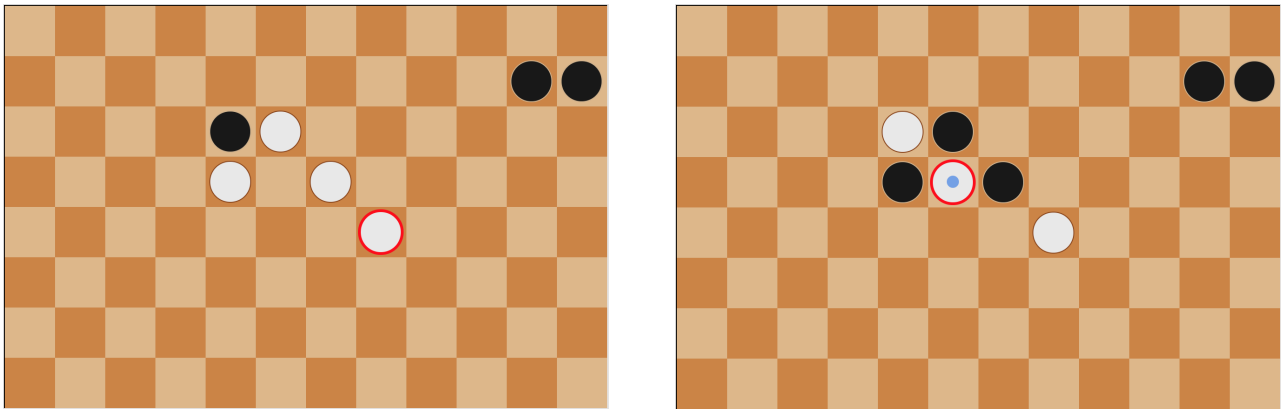
Ultimately, within the field of game-playing, researchers have understood that tree search and explore techniques are not sufficient for optimal performance on games of a certain complexity; Chess is simple enough that a traditional algorithm can reach grandmaster levels, but Go is not [1].

Since Pathwayz's game tree size fits between these two, we determined that some combination of traditional tree searching and unsupervised learning would allow PAI to achieve a high level of performance against talented players.

III. METHODS

To simulate the game tree, we rely on a state-based model. Each state describes the current player and the current board layout. The current player is either 'w' (white) or 'b' (black). The current board layout is an 8x12 matrix, where every element is either '-' (empty square), 'w' (white regular piece), 'b' (black regular piece), 'W' (white permanent piece), or 'B' (black permanent piece). Our GUI uses blue dots to signify permanent pieces. Our GUI also marks the most recent move with a red outline to provide a smooth UX.

The starting state is player 'w' (white always goes first) and an empty board (matrix filled with '-'). On a given turn for player 'w', a legal move is to place a 'w'



(a) Board layout after white places a 'w' piece.

(b) Successive board layout after black places a 'W' piece.

Fig. 3: Board and state model

(white regular piece) or 'B' (black permanent piece) on any '-' (empty square). Similarly for player 'b', a legal move is to place a 'b' (black regular piece) or 'W' (white permanent piece) on any '-' (empty square).

In the successor state, the current player becomes the other player ('w' or 'b'). The board layout includes the new piece. If the new piece was a permanent piece ('W' or 'B'), then any adjacent regular pieces switch colors.

Any and all features can be extracted from the current player and current board layout. These characteristics constitute the minimal viable representation of a game state. The following algorithms provide different ways of exploring this state-based model, with various advantages and disadvantages.

A. *Reflex Models*

- **Random** PAI randomly picks one move from its set of legal moves.
- **Baseline** PAI considers every legal move and chooses the one that maximizes its longestPath: the number of columns that PAI traverses in a contiguous path. The baseline essentially makes decisions based on this single feature at depth zero. Performing better than baseline more than 50% of the time assures us that a policy is a reasonable player.

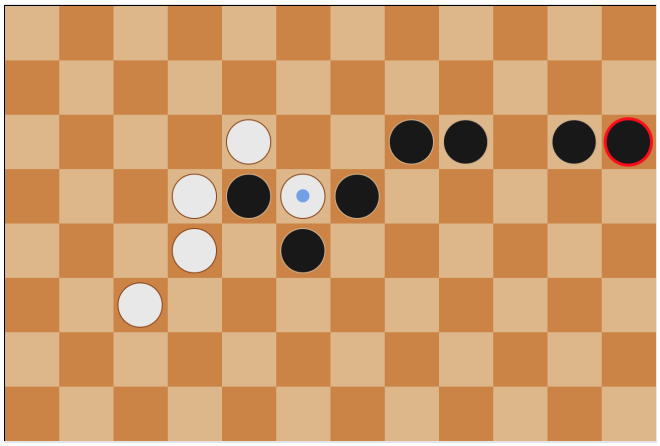


Fig. 4: White's longest path is 4, and black's is 5 (Note: that the two unconnected pieces do not contribute)

- **Advanced (a.k.a. Adversarial) Baseline** PAI considers every legal move and chooses the one that maximizes myLongestPath minus 0.4 times yourLongestPath. Play improves significantly as PAI now acts adversarially, both advancing its own path and blocking its opponent's path. The advanced baseline can consistently beat one of our three group members; performing better than advanced baseline more than 50% of the time assures us that a policy is a good player.
- **Dumb PAI** PAI splits the board into 3x3 grids. Every possible 3x3 grid acts as a feature. Each grid location can be '-', 'w', 'b', 'W', 'B', or 'None' (board edge), so there are more 2 million possible grids. We weighted these features through TD-learning.
- **Smart PAI** PAI operates based on domain specific features. We started with the two features from advanced/adversarial baseline (myLongestPath and yourLongestPath), and eventually built up to 440 total features. We weighted these features through TD-learning.

B. *Random Sample Models*

- **Monte Carlo Search** PAI uses a basic Monte Carlo search with 10,000 depth charges from the top ten moves, chosen via our evaluation function. MCS operates without defined features or weights, merely sampling end states to evaluate potential moves; PAI picks the move with the highest percentage of wins.
- **Monte Carlo Tree Search** PAI uses the more advanced Monte Carlo tree search with 250,000 depth charges per turn to generate slices of the game tree and explore new branches more intelligently. Unlike MCS, MCTS keeps track of which nodes it has previously visited, and back-propagates the result from each depth charge across the sequence of explored nodes. MCTS makes exploration decisions based on an upper confidence bound function that weighs a node's utility against the number of times that it and its parent have been visited.

C. *Adversarial Search Models*

- **Minimax** PAI uses a depth-limited minimax algorithm with alpha-beta pruning. Despite these attempts to limit complexity, PAI moves incredibly slowly at a depth of 2 in opening game (when breadth is the maximum 192 legal moves). Therefore, we added depth-varying capabilities. PAI begins at depth 0, then increases depth as the board fills up (i.e. as breadth decreases).
- **Expectimax** PAI uses a depth-limited expectimax algorithm. PAI considers the expected value of its opponent's possible moves instead of its opponent's optimal move. Because there are so

Fig. 5: Narrowing beam minimax example

many terrible moves in Pathwayz (e.g. giving your opponent a permanent piece), we only averaged over the top ten moves, chosen via our evaluation function.

- **Narrowing Beam Minimax** If PAI notices that the other player is one move away from winning, PAI follows the same Minimax policy as above, with a depth of 2. (PAI retains reasonable computation time at this stage because the only way a player can be one move away from winning is if many squares are filled and the breadth is reduced.) Otherwise, PAI uses a narrowing breadth to allow for greater depth. PAI simulates its top ten moves, its opponent's top five moves, then its own best move.

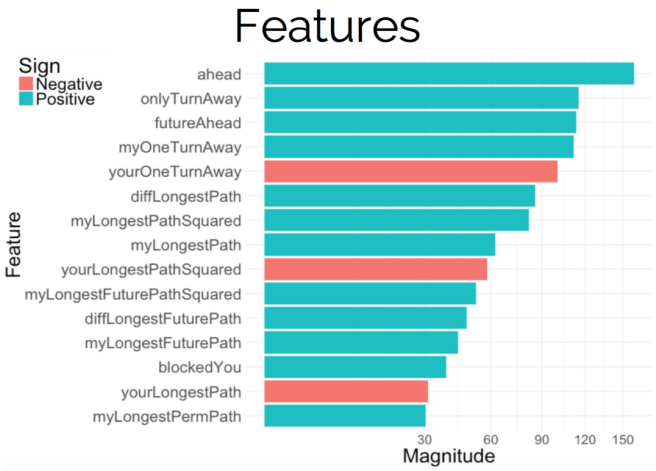


Fig. 6: Top 15 smart features after TD Learning

D. Features

We hand-picked features and initialized weights based on domain knowledge. We came up with various ways to measure the following for each player:

- Path length
- Path permanence
- Number of columns
- Number of permanent pieces
- Number of regular pieces with X empty neighbors
- Number of pieces you can flip at once

Our breakthrough was establishing lookahead features. By storing the left and right frontiers of paths, we were able to efficiently estimate:

- Future path length
- If a path is blocked
- If someone is one turn away from winning

We also included squared, interaction, and indicator features for most of the above. We created an evaluation function based on normalized features and weights. We also created a utility function of 1e6 based on the winner. (When the utility function was $\pm\infty$, PAI would place randomly if it couldn't prevent the other player from winning. However, we still wanted PAI to place optimally in case the other player made a mistake.)

Our primary PAI used for the first 5,000 games of TDL included 52 features. Our final PAI used for another 5,000 games of TDL included additional path measurements and interaction features; we also created versions of each feature for the early, mid, and end-game stages, resulting in 440 total features.

E. TD Learning

PAI employed temporal distance learning in over 10,000 games to learn weights for its domain-specific features. Weights updated every two plies. We provided a large positive/negative end-game reward for winning/losing. We wanted to make sure PAI didn't overfit to a local minimum, so we used exploration probabilities. PAI explored among its top ten moves based on a random epsilon (highest scored move if $\epsilon < \frac{1}{2}$, second highest if $\epsilon < \frac{3}{4}$, third highest if $\epsilon < \frac{7}{8}$, etc.). We also wanted to regularize weights, so we created an automated training pipeline.

If PAI wins too many games while training, it becomes overconfident, increasing all of its feature weights (including ones that should be negative). After winning at least 60% of games, PAI's opponent adopts PAI's current weights, which ensures that PAI is consistently challenged at an appropriate skill level.

Conversely, if PAI loses too many games, it becomes pessimistic, decreasing all of its feature weights (including ones that should be positive). After winning

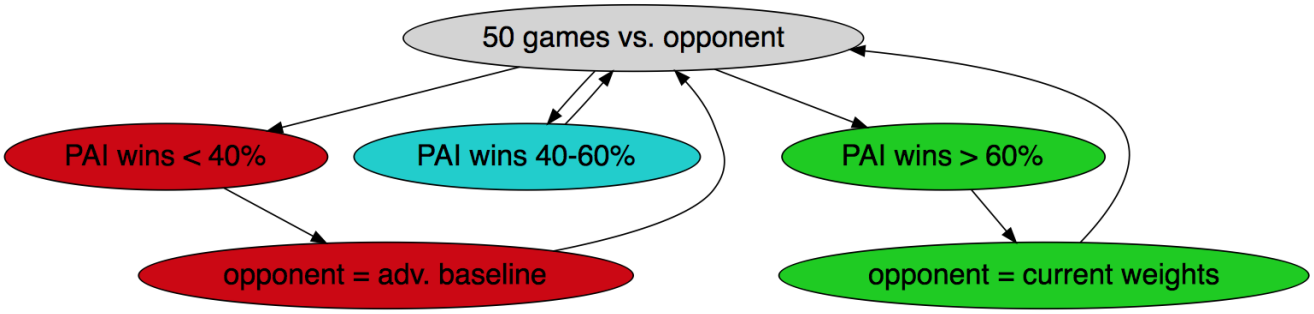


Fig. 7: Automatic TD Learning Pipeline

under 40% of games, PAI switches to play against advanced baseline, which it can learn to consistently beat. PAI subsequently builds back confidence, regularizing features before returning to a more challenging opponent.

We put PAI, with 52 features, through this pipeline for 5,000 games. We then fine-tuned weights manually and against beam minimax. After developing more features, we put PAI through this pipeline for another 5,000 games. Once again, we fine-tuned weights against beam minimax. Our final PAI consists of 440 domain-specific features, trained in over ten thousand games via TDL, and applied in a narrowing beam minimax algorithm.

F. Final PAI

PAI achieved its ultimate form by applying the 52 features it trained in TDL to a narrowing beam minimax algorithm. Similar to the earlier implementation, PAI searches the game tree at several depths, achieving higher performance by limiting the search space through a narrower breadth. PAI searches its top 10 best moves, its opponent's five best moves and finally its best move after that.

Our final PAI consists of 440 smart features, with weights trained in over ten thousand games through Temporal Difference Learning, applied in Narrowing Beam Minimax. PAI the game's inventor in games.

IV. RESULTS

To compare the performance of various algorithms, we placed them in a round-robin tournament. The PAIs all played one another out of 20 games (a total of 720 games). Win rates are presented down the columns, and loss rates are across the rows. Although there is no known first or second-player advantage, we had each AI play black and white the same number of times to ensure an even match-up.

	Baseline	Adv. Baseline	TDL (52 features)	TDL (440 features)	MCTS	Minimax	Beam Minimax	TDL Beam Minimax (52 features)	TDL Beam Minimax (440 features)
Baseline		60%	100%	95%	70%	80%	100%	100%	100%
Adv. Baseline	40%		80%	80%	40%	75%	90%	95%	95%
TDL (52 features)	0%	10%		85%	0%	35%	40%	90%	90%
TDL (440 features)	10%	20%	15%		25%	65%	85%	85%	90%
MCTS	0%	30%	100%	75%		80%	90%	100%	95%
Minimax	0%	30%	65%	35%	20%		90%	100%	75%
Beam Minimax	0%	10%	60%	15%	10%	10%		85%	55%
TDL Beam Minimax (52 features)	0%	0%	10%	15%	0%	0%	15%		
TDL Beam Minimax (440 features)	0%	5%	10%	10%	5%	25%	45%		

Fig. 8: Bracket play results

First, let's acknowledge some of the less successful algorithms. MCTS, while able to beat baseline, did not perform well against other players. We think MCTS's poor performance stems from a similar issue as expectimax's poor performance: random moves in Pathwayz are often terrible (e.g. giving your opponent a permanent piece), so random sampling is a weak predictor. Minimax was much better than MCTS, but struggled to contend with the strongest players. As a reminder, we implemented a depth-increasing minimax because the initial game breadth was too large for a depth greater than one. Consequently, minimax is essentially a reflex model in the early game, and suffers accordingly.

Reflex TDL has only a slight advantage against Beam Minimax. Of course, that it has any advantage at all implies that our choice of trained features has a significant impact on performance. This is further reinforced by TDL Beam's crushing defeat of Beam Minimax.

TDL features had a significant impact on performance. The two reflex models trained via TDL performed much better than advanced baseline. TDL with 52 features manages to scrape out a 60% win rate against our initial beam minimax; a reflex model defeating an adversarial search model is a remarkable achievement. We think the TDL with 440 features suffers from overfitting. It trained against itself and beam TDL minimax, so is optimal against those players, but needs more tuning against other players. The TDL with 52 features is adaptable enough to regularly defeat humans. In fact, this reflex version of PAI beat a random sample of humans 33-0.

TDL beam minimax was the winner in this tournament. TDL beam minimax with 440 features defeated TDL beam minimax with 52 features, but the latter is more generalizable than the former. All in all, combining TDL features with a lookahead algorithm led to a smart, versatile player. Against our team, PAI beats Nikhil Prabala 5/5 times, Michael Tucker 4/5 times, and Louis Lafair 3/5 times.

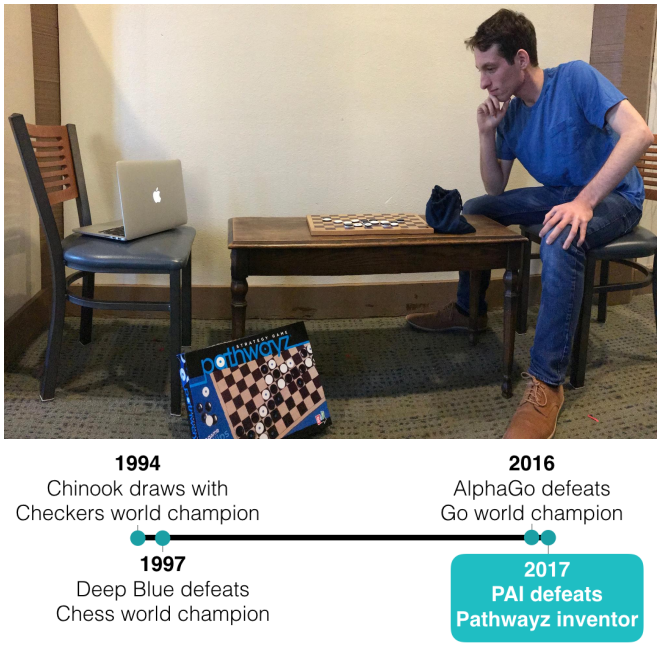


Fig. 9: PAI defeats Louis Lafair

V. DISCUSSION

A. Error Analysis

It is worth noting that several of our algorithms do not consistently beat the advanced or normal baseline in every game. We attribute this to a combination of factors.

First, our sample size of 20 games does not give us statistical significance, meaning that our results could vary greatly from the true performance of each algorithm. (20 was chosen due to time constraints running the slower algorithms.) We expect that after roughly 100 games, we would begin to converge on the true performance.

Second, several of our algorithms choose randomly among equally good moves, as determined by a given heuristic. A standard of play is consistent for a given algorithm, but its exact style or set of tactics can vary widely from game to game. At a high enough level, these differences could make or break victory, leading to less reliable outcomes. However, we do not believe that this factor biases the data in such a way that obscures general comparisons between our implementations.

Third, we could be overfitting our weights, in particular with our 440 TDL features. (We certainly overfit when we tried our more than 2 million dumb grid features.) During training, PAI may have become too accustomed to certain playstyles that don't generalize to other algorithms or humans. Similarly, our minimax models, which expect strong opponents, can sometimes be beaten by the relatively simple advanced baseline tactic.

B. Player Advantage

A first-player advantage exists in many two-player strategy games. In Go, the second player receives a compensation to remove first player advantage. In Chess, the first player has a winning percentage of around 55%. Winning percentage is calculated as:

$$\frac{\text{wins} + \frac{1}{2}\text{draws}}{\text{Total Games}}$$

Given the lack of historical data, there is no known first player advantage in Pathwayz. Some have posited a first player advantage.

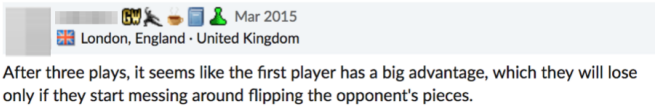


Fig. 10: Comment on BoardGameGeek.com about first-player advantage in Pathwayz

We decided to see if the theory holds, and to what extent. We played PAI in 1000 games against itself. White won 487 games, black won 499 games, and they drew 14 games. In other words, the first player had a

win rate of 49.4%. If anything, there seemed to be a minor second player advantage, though it may decrease with more simulations or more advanced AIs. Black's advantage was so small that we can assume (until a larger game database exists) that there is no significant player advantage.

C. Surprises

PAI, as the first Pathwayz AI, can teach us novel approaches to Pathwayz. There have been several surprises in its playstyle. Most human players are conservative, waiting to flip until middle or end game. PAI, on the other hand, is aggressive, often flipping in the first several moves. PAI tries (often successfully) to get an early lead so that it can play offensively instead of defensively at end game.

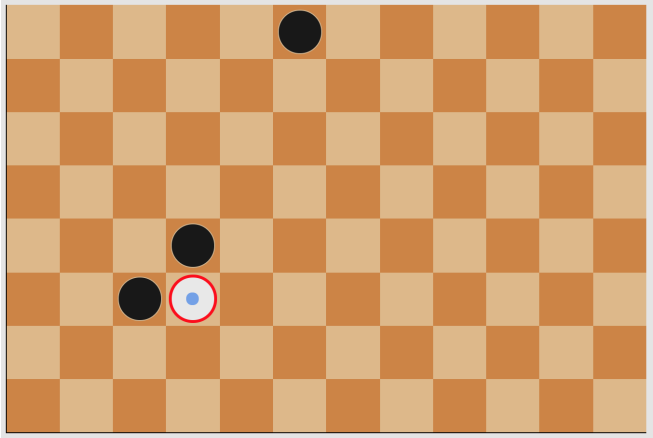


Fig. 11: PAI (black) plays permanent piece, aggressively flipping at start of game

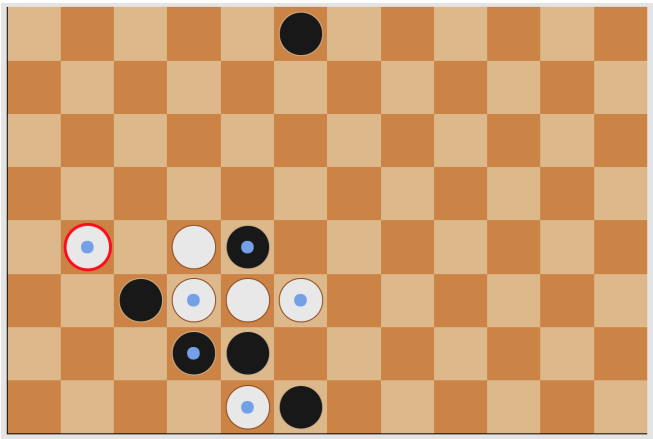


Fig. 12: Later in the same game as 11, PAI has now played permanents in four of its first six moves

Also surprising is PAI's ability to block flips. PAI will sometimes play in unconventional spaces simply to block an opponent's flip.

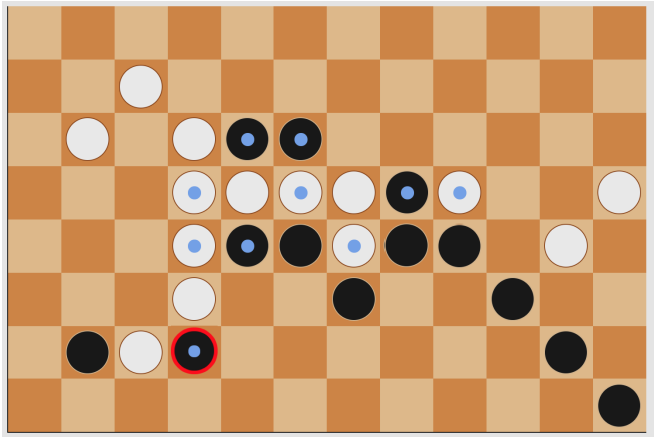


Fig. 13: Louis Lafair (white) just moved, so it's PAI's turn (black)

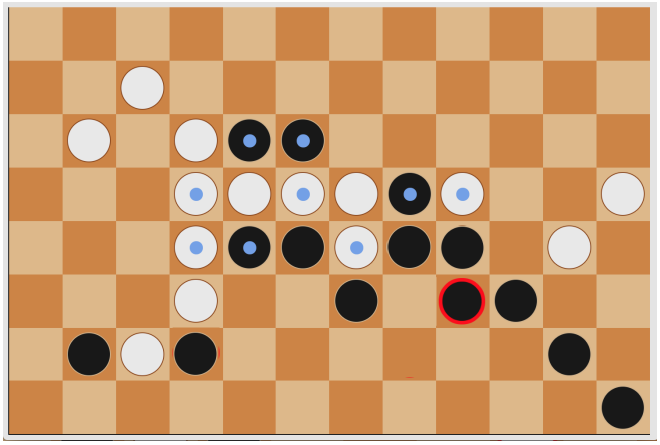


Fig. 14: PAI (black) strategically places to block a dangerous flip

PAI will do whatever possible to prevent its opponent from winning, even, at times, flipping its own pieces.

Continuing to play and develop PAI will reveal more novel strategies.

D. Future Work

Moving forward, there are a few optimizations that we feel would make PAI even stronger. First, we want to train PAI against human players we have already built the infrastructure to begin doing so. Second, we could try combining Monte Carlo Tree Search with our trained evaluation function, which may improve

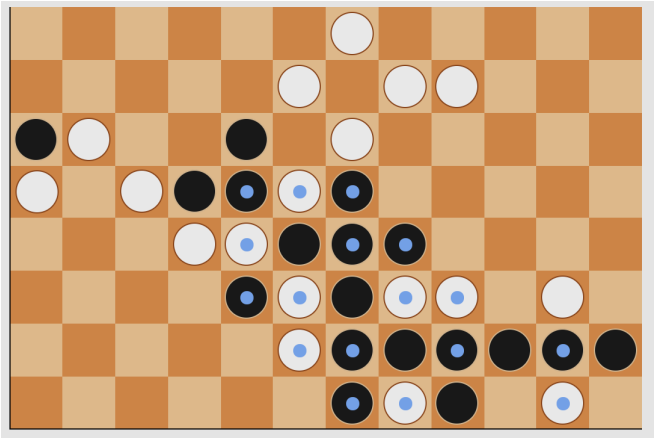


Fig. 15: Louis Lafair (black) is one turn away from winning

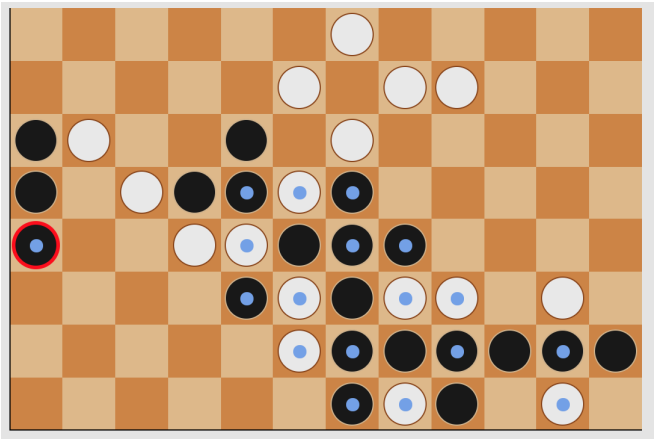


Fig. 16: PAI (white) makes a sacrifice, flipping its own piece to prevent a winning flip next turn

PAI's foresight. Third, we want to reduce (and potentially add) features, settling somewhere between 52 and 440. Finally, employing neural nets would allow for more complicated feature evaluations, potentially helping PAI learn more nuanced strategies.

PAI in its current form is a powerful player, but not unstoppable. Moving forward, we intend to change that. PAI's fast improvements demonstrate potential for super-human play.

References

[1] Burmeister, Wiles, The Challenge of Go as a domain for AI research: A comparison between Go and Chess. Proceedings of Third Australian and New Zealand Conference on Intelligent Information Systems, 1995, ieeexplore.ieee.org/document/705737/ Accessed 15 December 2017.

[2] Campbell, Hoane, Hsu, Deep Blue, Artificial Intelligence, Vol. 134, No 1, 2002, sciencedirect.com/science/article/pii/S0004370201001291#! Accessed 15 December, 2017.

[3] DeepMind, Mastering the game of Go with deep neural networks and tree search. Nature, Vol. 529, 2016, storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf. Accessed 15 December, 2017.

[4] HSU, IBM's Deep Blue Chess grandmaster chips. IEEE Micro, Vol. 19, No. 2, 1999, ieeexplore.ieee.org/document/755469/ Accessed 15, December, 2017.

[5] Miller, Andersen, Liu, Karlin, Popovic, Evaluating Competitive Game Balance with Restricted Play. Evaluating Competitive Game Balance with Restricted Play. 2012. aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/viewFile/5470/5692 Accessed 15, December, 2017.

[6] "Pathwayz Comment Page" BoardGameGeeks, 15 Dec. 2017, boardgamegeek.com/boardgame/164593/pathwayz/ratings?comment=1

[7] Pathwayz Home, Pathwayz, 15 Dec, 2017, pathwayzgame.com/

[8] Rosenbloom, A world-championship-level Othello program, Artificial Intelligence, Vol. 19, No. 3, 1982, www.sciencedirect.com/science/article/pii/0004370282900030 Accessed 15, December, 2017.

[9] Tesauro, Temporal Difference Learning and TD-Gammon. Communications of the ACM, Vol. 38, No. 3, 1995, cling.csd.uwo.ca/cs346a/extra/tdgammon.pdf