Multi-Minimax: A New AI Paradigm for Simultaneously-played Multi-Player Games

Nicolas Perez[1] and B. John Oommen[2] [⋆]

School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6.
[1]nickperez@cmail.carleton.ca,[2]oommen@scs.carleton.ca

Abstract. The best reported methods for turn-based multi-player game playing AI algorithms include Max^n, Paranoid and Best Reply Search. All of these methods make decisions by modelling the game by assuming that there is some predetermined play ordering for the players. While this is meaningful for ordered turn-based games, there are a host of scenarios where the players need not be constrained to make their moves in such a manner. Little research has been done for turn-based games of this kind such as financial games that involve buying and selling on the stock market in no specific order[1]. In this paper, we shall present and test a new algorithm for multi-player game playing on a game which does not require a fixed sequential play ordering. The game that we have used to demonstrate this is the multi-player Snake Game, also referred to as a "Light Bike" game which is a turn-based game requiring simultaneous moves at every turn. Our newly-proposed scheme, the Multi-Minimax, along with the Added Pruning method, performs better when compared to the similar AI strategies examined in this paper. Additionally, among all the algorithms that did not use the proposed pruning, Multi-Minimax performs the best. We can conclude that, at the least, under certain conditions in the area of multi-player game playing AI, similar results can be replicated with these newly proposed Added Pruning and Multi-Minimax methods. As far as we know, the results presented here are of a pioneering sort, and we are unaware of any comparable results.

Keywords : *Alpha-Beta Pruning, Best Reply Search, Max^n, Paranoid, Minimax, Multi-Minimax, Multi-player Games, Game Tree Pruning*

1 Introduction

Multi-player games have to be tackled in AI with techniques that are distinct from those used in two-player games. This is because the heuristic function used for any game can yield multiple values for the various players, and alternatively, could lead to a *vector* of heuristic values. Such a vector implicitly prohibits Minimax, Alpha-Beta search and a tree pruning strategy. An alternative strategy,

[⋆] *Chancellor's Professor*; *Life Fellow: IEEE* and *Fellow: IAPR*. This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway.

[1] For games with shared resources (e.g., financial games) or simultaneously-played move games, one could alternatively consider multi-player AI algorithms to be those that treat the game with each opponent as a separate game. This is currently open.

adapted by the Best Reply Search (BRS) would be to utilize a single heuristic function and to consider all the opponents as possible players at the next time instant. While this allows for Minimax and the consequent pruning strategies, they have all been specifically used for games in which there is a sequential ordering for the players' moves. However, the scenario is quite different when the players can play simultaneously at every time instant or in a random fashion.

This is the arena in which we operate, and the aim of this paper is to consider multi-player games in which the players are not constrained to play in any specific order. Games of this sort are typical in the stock market and financial sector, where a buyer/seller does not have to wait for the others. The aim of this paper is to demonstrate that the Perspective player can invoke the BRS as his strategy without the other players being aware of it. By resorting to such a technique, the Perspective player can make meaningful choices that yield a superior win rate and still permits him to search to greater depths in the search tree by resorting to a Minimax paradigm and alpha-beta pruning. Apart from demonstrating this, we benchmark such simultaneously-played multi-player games.

2 Description of Problem Domain

To demonstrate the power of our technique in a *prima facie* manner, we shall use it to play the game "Light Bike" (please see Figure 1), which is a popular form of an arcade-style multi-player game, played on various platforms. It involves players who can move in one of 4 directions on a 2D grid, where each player leaves behind themselves a wall which cannot be passed through by any player. The goal for each player is to be the last surviving player.

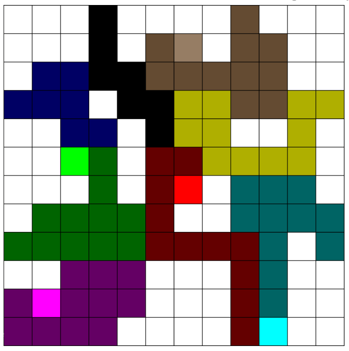

Fig. 1: An example of a game of "Light Bike" with 8 players.

Each player must move to one of its adjacent tiles (excluding tiles diagonally adjacent) at every turn, and all the players make their moves simultaneously and within a fixed time limit. Players are eliminated if they occupy a space on the grid that has been or is currently occupied by another player, or if they move off the edge of the screen. Under certain circumstances, a player is inevitably eliminated from the next turn independent of what move is made due to him being "boxed in" by surrounding walls. Effective strategies involve players trying to box other players within a small area while they themselves are not boxed in by other players' walls or even their own walls. In Figure 1, yellow, black and dark blue players have already been eliminated. Green, red, magenta, brown and turquoise players are alive. Magenta, with two possible spaces, faces elimination.

3 Multi-Player Game Strategies and Approaches

This section will describe competitive solutions to the Multi-Minimax algorithm. We only consider search-tree based algorithms for multi-player turn based games. They involve the Perspective player (the root of the tree) to search through combinations of a certain move order which may or may not truly represent the move order of the game, and to try and predict which next move in the game would be most effective. When it is the Perspective player's turn, he will search through nodes (each representing a game state) up to a specified maximum depth in the tree, and use a heuristic function to evaluate leaf nodes in the tree. The evaluations of leaf nodes are recursively carried up to the root node based on the algorithm's specified strategy for deciding which evaluations are carried up each node. Thereafter, he plays the best-evaluated next move dictated by the root.

The Minimax principle has been shown to be an effective scheme for playing two-player games [2, 7, 9]. Compared to traditional two-player game playing, multi-player environments, through the addition of other self-interested agents, introduce a range of new complications and challenges. These include:

– Any single player's gain need not lead to an equal loss among the opponents;
– Player coalitions can arise, even in games with only a single winner;
– The board state can change more between the Perspective player's moves;
– A single-valued heuristic is not always sufficient to appraise the game state;
– Established pruning schemes, e.g., the alpha-beta, are not always applicable;
– The computation is exponential with respect to the number of players.

Despite these challenges, due to the historical success of Minimax with alpha-beta pruning in a wide variety of domains, substantial efforts have been dedicated to extending it to multi-player environments, with varying levels of success [1, 4–7]. We now detail a number of the more well-known of these techniques, specifically the Paranoid, Max^n, and the BRS.

3.1 The Paranoid Algorithm

The intuitive extension of the Minimax technique to multi-player games results in what is commonly termed the "Paranoid algorithm" [1, 4, 6]. This approach requires the fewest changes from the well-known Minimax algorithm [7, 9, 2]. As in the Minimax, the Paranoid technique retains a single value at each node representing the heuristic value *h(x)* for the Perspective player. Being a multi-player scenario, rather than two player, each level of the game tree represents a different player's turn, and so there will be multiple opponent turns in between each of the Perspective player's turns. The Paranoid algorithm handles this by treating every opponent's turn as a Min node [4]. Thus, for a three-player game, the Paranoid algorithm could be referred to as Max-Min-Min, and for a four player game, Max-Min-Min-Min. A sample game tree for an arbitrary evaluation function, is presented in Figure 2 for the Paranoid algorithm. The values indicate how "good" each node (board position) is for the Perspective player.

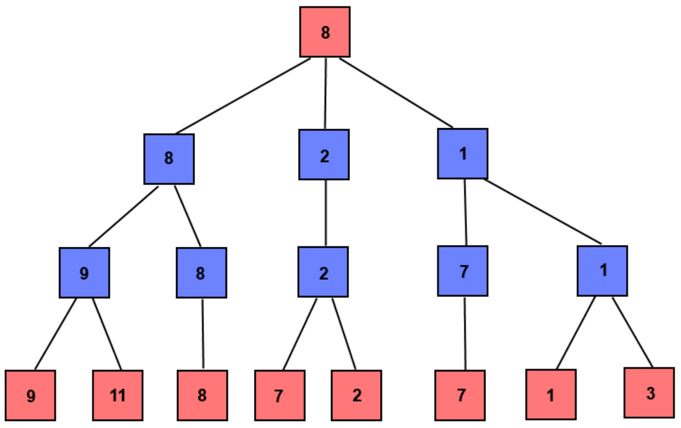

Fig. 2: A Paranoid Tree, with the red nodes bing MAX, and the blue nodes MIN.

Since all the Perspective player's opponents are "minimizing" (and not maximizing their own gains), the Paranoid algorithm, naturally, treats all players as a *coalition* against the Perspective player [6]. The algorithm even predicts that opponents will take moves operating under the assumption that *other* opponents will take actions leading to even greater minimization of the Perspective player's score. Thus, opponents not only exclusively target the Perspective player, but will, in fact, actively work together against him [6].

The Paranoid algorithm suffers from a glaring drawback. While it could be considered the "safest" approach to the game, it is unreasonable to work with the assumption that opponents in a multi-player game will solely work in a coalition, even to their own potential detriment. It thus has a tendency to consider unrealistic game states, which could potentially lead to bad play, particularly when there are dramatic shifts in board positions between moves [6].

Since it maintains a single heuristic value, the Paranoid algorithm retains the benefits of alpha-beta pruning, which is not the case for all multi-player techniques. It thus outperforms other, more realistic strategies due to achieving improved lookahead [1]. Despite this, it can only produce cuts at at boundaries between Max and Min nodes, and never between individual Min layers. Thus, less total pruning will occur in a Paranoid tree than in a Minimax tree.

3.2 The Max^n Algorithm

While the Paranoid algorithm is the most intuitive extension of Minimax to multi-player games, and the simplest to implement, a competing algorithm, called the Max^n algorithm, is the natural extension of Minimax principles to N-person games [5]. The basic philosophy of the Minimax algorithm is not based on minimizing a specific player's score, but instead on maximizing the AI player's score [7, 9, 2]. For a two-player, zero-sum, combinatorial game for which the Minimax algorithm was originally developed, these two functions are naturally identical, the extension of which is the Paranoid algorithm [4]. However, the Max^n algorithm operates on the more reasonable assumption that players will seek to maximize their own scores, without consideration for other opponents [5].

Rather than the heuristic function *h(x)* returning a single value, as is the case with the Minimax and Paranoid algorithms, the heuristic function for the Max^n algorithm returns a *tuple* of values of size N, where N is the number of players [5]. The N^{th} value, traditionally, corresponds to the N^{th} player, where the first player is the Perspective player, and where subsequent opponents are numbered in their turn order, beginning from the Perspective player. At the i^{th} player's turn, he is assumed to choose the move that provides the maximum value in position i in the tuple, and, similar to the Minimax or Paranoid algorithms, this value is passed up the tree, until, eventually, a path is chosen for the Perspective player at the root [5]. Figure 3 shows a sample Max^n tree after expansion of all the leaf nodes, with the values being passed up to the root. The values associated with each node represent the tuple returned by the algorithm's heuristic function.

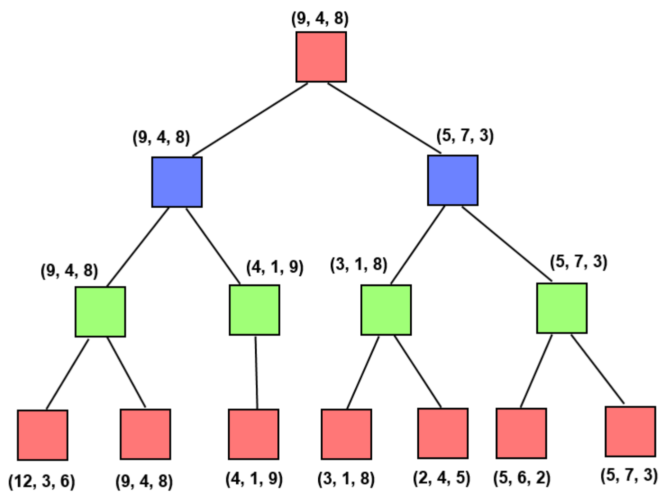

Fig. 3: Sample Max^n Tree in which each color represents a different player.

There is a clear benefit of using it over the Paranoid algorithm [6], as the more "relaxed" pressure on the Perspective player can take advantage of the various opportunities that may have been overlooked by "coalition of opponents" [4].

Despite a more realistic model of play, as the Max^n algorithm makes use of a tuple of values rather than a single integer value for the results of the heuristic, it can't make use of alpha-beta pruning. It can only make use of less-effective pruning techniques described in [3, 14–16], and is also useful in tie breaking [4].

3.3 Best Reply Search

The Paranoid and Max^n algorithms remained the standard for deterministic multi-player games. However, more recently, an algorithm named the Best Reply Search (BRS) has been introduced, which can, in some cases, significantly outperform both of them [1]. In the case of the BRS, all opponents are again considered to operate as in a coalition, as in the Paranoid algorithm, but between each of the Perspective player's turns, *it allows only a single opponent to act* [1]. The opponent who is allowed to act is the one who has the most minimizing move, in relation to the Perspective player, at this point in time, or the "Best Reply". In essence, the scheme pretends that all opponents are not simply a coalition, but that the coalition represents a single player with significantly more resources available than the Perspective player. Figure 4 shows a single level of a BRS tree where the minimum of all opponent turns is being selected.

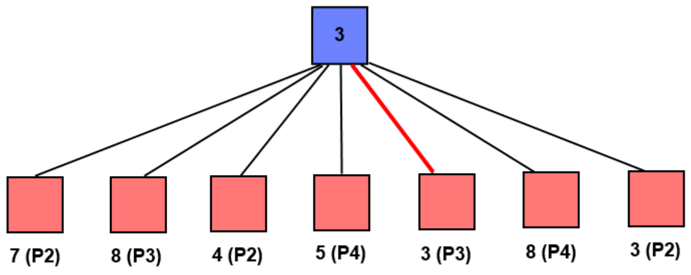

Fig. 4: The operation of a single level of the Best Reply Search. The scores that are reported have the opponent's player number listed next to them (in parenthesis) to assist in the clarification.

The glaring drawback of the BRS algorithm is that it considers illegal move states while searching. This is certainly a serious drawback, and it limits the games to which the BRS can be applied [1]. It can only be applied to those games where it is *meaningful* for players to act out of turn, and performs best when the board state does not change dramatically in between turns [1]. Whenever the game state changes significantly between turns, there is a serious risk of the BRS arriving at a model of the game which is significantly different from reality.

In cases where it can be applied, the BRS has many benefits over the Paranoid and Max^n algorithms, and often dramatically outperforms them [1]. As can be intuitively observed, when all opponents are considered to be a single entity, the game is modeled as a two-player game played using the Minimax algorithm.

Alpha-beta pruning and all other two-player Minimax improvements can be applied with even less restrictions than in the Paranoid algorithm [2, 7, 9]. As it simplifies things to work with a model analogous to a two-player game, the BRS also allows better look-ahead for the Perspective player than either the

Paranoid or the Max^n schemes. These are significant factors in the performance of the BRS over the Paranoid and Max^n in games such as Chinese Checkers [1].

4 Motivation and Proposed Solution

The goal of this paper is to find a way to significantly increase the search-depth for the Perspective player in a search-tree algorithm for playing multi-player games in which the players are allowed to move simultaneously. One drawback of previously-mentioned solutions is that their asymptotic run time is exponential with respect to the number of players and the depth of the search-tree. If an effective solution is found for one game, it could then be possibly applied to other non-turn based simultaneously-played multi-player games. While we have proposed such a solution, we have also demonstrated its power using the game "Light Bike". This game was chosen due to its dependence on effective players having to be able to search a large number of moves ahead.

4.1 The Proposed Solution: Multi-Minimax

The method we propose is the so-called Multi-Minimax algorithm described formally in Figures 5, 6 and 7. It can be seen as the BRS but with a very aggressive pruning method which makes it so that minimizing players cannot alternate taking turns against the maximizing player. Given the current game state, the maximizing player plays $n-1$ different games against each of the $n-1$ opponents and makes a move assuming that depending on the next move the maximizing player makes, the maximizing player will be playing the rest of the game only against the best opposing player for that move. To describe the Multi-Minimax algorithm we invoke the Minimax algorithm which is used as a subroutine for Multi-Minimax [2, 7, 9]. Observe that when n is the number of players, b is the branching factor and d is the maximum number of game rounds (the number of times all players have each taken one turn), the Multi-Minimax runs in $O(nb^d)$ worst case time. On the other hand, Paranoid, BRS and Max^n run in $O(b^{dn})$, $O(n^{\frac{d}{2}}b^d)$ and $O(b^{dn})$ worst case times respectively [1, 4–6].

4.2 Added Pruning Method

Our proposed Added Pruning restricts the next move that each player can explore along the search tree. This pruning forces him to explore only the next move that is in the same direction as his previously-explored move. If he is not able to explore a move in this previously-explored direction, he would be able to explore all possible valid moves given by the default production system.

One benefit to using this method is that players are less likely to search meaningless paths where they eliminate themselves from the game, for example in "Light Bike", by boxing themselves in with their own walls. It also emphasizes exploring moves that are significant since, usually, the most significant moves in the game are when players move to a grid space that is right next to a wall to box in an opposing player. Such a pruning allows a larger look-ahead and to thus explore a larger distance across the board and explore more moves for boxing in players. Additionally, the Added Pruning allows players to be less likely to pointlessly search through multiple different move sequences where they could reach the same grid space in the same number of moves considering all the move sequences that yield similar effectiveness. The downside to possibly missing exploring significant moves due to the Added Pruning does not outweigh the benefits previously described.

Algorithm 1 Multi-Minimax

```
1: procedure MULTI-MINIMAX($maxPlayer, minPlayers, depth, gameState$)
2:     $moveToMake \leftarrow 0$
3:     $maxMove \leftarrow -\infty$
4:     $\alpha \leftarrow -\infty$
5:     for each child of maxPlayer do
6:         updateGameState($gameState, child$)
7:         $minMove \leftarrow \infty$
8:         $\beta \leftarrow \infty$
9:         for each opponent in minPlayers do
10:            $minMove \leftarrow$ min($minMove$, MINIMAX($child, opponent, depth - 1, \alpha, \beta, FALSE, gameState$))
11:            $\beta \leftarrow minMove$
12:            if $\alpha \geq \beta$ then
13:                break
14:            end if
15:        end for
16:        if $minMove \geq maxMove$ then
17:            $moveToMake \leftarrow child$
18:            $maxMove \leftarrow minMove$
19:            $\alpha \leftarrow maxMove$
20:        end if
21:    end for
22:    return $moveToMake$
23: end procedure
```

Fig. 5: Pseudocode for the Multi-Minimax algorithm with alpha-beta pruning which also uses Minimax as a subroutine [2, 7, 9].

4.3 Implementing the Formalized Method

We implemented the "Light Bike" game using the Python language invoking the Multi-Minimax, BRS, Max^n and Paranoid algorithms [1, 4–6]. The heuristic used for all these was as follows: At each node, if there were no more valid moves (i.e., which resulted in the player surviving for the next turn) for the opposing player(s) but there is at least one valid move for the evaluated player, we returned "infinity", otherwise we returned the depth of the node in the tree.

The implementations for all the algorithms except for Max^n used immediate pruning at the root node and alpha-beta pruning [2]. Max^n could only make use of immediate pruning at every node and tie breaking by minimizing the sum of the score of opposing players [4, 3, 14–16]. Each algorithm made use of

Algorithm 2 Minimax

```
procedure MINIMAX($maxPlayer, minPlayer, depth, \alpha, \beta, max?, gameState$)
    if $depth = 0$ or no more valid moves then return heuristic value
    end if
    if $max?$ then
        $maxMove \leftarrow -\infty$
        for each child of maxPlayer do
            updateGameState($gameState, child$)
            $maxMove$ ← max($maxMove$, MINIMAX($child, minPlayer, depth - 1, \alpha, \beta, FALSE, gameState$))
            $\alpha \leftarrow$ max($\alpha$, $maxMove$)
            if $\alpha \geq \beta$ then
                break
            end if
        end for
        return $maxMove$
    else
        $minMove \leftarrow \infty$
        for each child of minPlayer do
            updateGameState($gameState, child$)
            $minMove$ ← min($minMove$, MINIMAX($maxPlayer, child, depth - 1, \alpha, \beta, TRUE, gameState$))
            $\beta \leftarrow$ min($\alpha$, $minMove$)
            if $\alpha \geq \beta$ then
                break
            end if
        end for
        return $minMove$
    end if
end procedure
```

Fig. 6: Pseudocode for the Minimax algorithm.

Monte-Carlo reordering of the production system at each node as well as pruning for game states where the Perspective player had been eliminated. Additionally, in our experiments we compared performances with and without the Added Pruning at each node (described in Section 4.2).

5 Results

Tests were done for the "Light Bike" game on a 12×12 sized board, comparing Multi-Minimax, Best Reply Search, Max^n and Paranoid algorithms, each of them with and without the Added Pruning described in Section 4.2. Each algorithm used iterative deepening with the only restriction of having 300 ms of real-world time to iterate [8]. All tests were ran on a machine with Linux and an Intel(R)

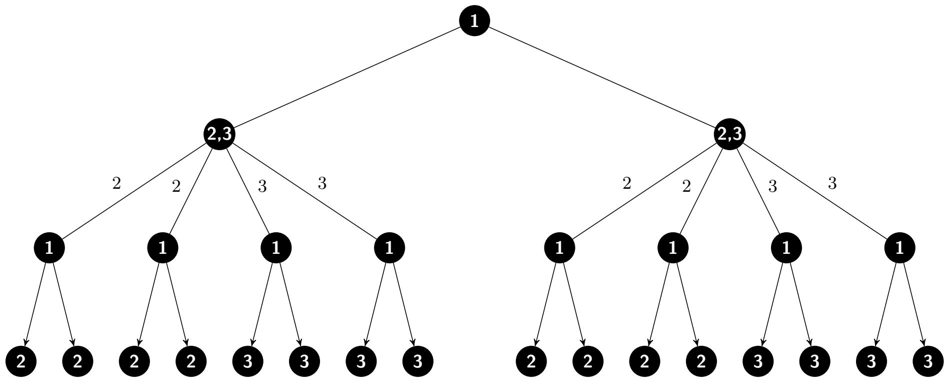

Fig. 7: A visual representation of the Multi-Minimax algorithm with 3 players and a branching factor of 2 for each player. Each node represents a game state and is labeled with the player(s) whose turn is next down the search tree. Each edge represents a player's move. Nodes with multiple players have their outgoing edges labeled to indicate which player moved.

Xeon(R) CPU E5-2600 v4 @ 2.00 GHz, which, as of writing this paper, has around the same processing power as the average personal computer.

Each Implemented Algorithm vs Two Randomly Moving Opponents			
Implemented Algorithm	**Average Depth**	**Win %**	**Win % w. Ties**
Max^n	6.86 ± 0.01	(49.09%, 51.86%)	(54.48%, 57.23%)
Max^n with Added Pruning	14.80 ± 0.04	(58.27%, 60.99%)	(63.75%, 66.39%)
Paranoid	9.51 ± 0.02	(57.27%, 60.0%)	(63.24%, 65.89%)
Paranoid with Added Pruning	18.39 ± 0.05	(65.44%, 68.05%)	(72.02%, 74.47%)
Best Reply Search	8.29 ± 0.02	(58.21%, 60.93%)	(64.76%, 67.38%)
Best Reply Search with Added Pruning	13.47 ± 0.05	(65.2%, 67.81%)	(71.61%, 74.07%)
Multi-Minimax	9.43 ± 0.02	(61.69%, 64.37%)	(68.21%, 70.76%)
Multi-Minimax with Added Pruning	19.29 ± 0.05	(73.07%, 75.49%)	(79.05%, 81.26%)

Fig. 8: The results of running each one of the implemented algorithms *vs* two randomly moving opponents. 5,000 tests were run for each algorithm. In the tables, P+ implies that the scheme has been enhanced with Added Pruning.

The results are given in Tables 8, 9 and 10. For one set of tests, each algorithm being tested would play against 2 randomly moving players which would play a random valid move (a move that wouldn't result in guaranteed elimination for that player's next turn) each turn so long as one existed. Another set of tests put the 8 different algorithms from the first set of tests versus each other in varying combinations of 4 and 6 player games. For the varying combinations of 4 and 6 player games, all 8 choose 4 and 8 choose 6 participant combinations were uniformly sampled. For testing purposes, a win for a player counted as being the last player standing. Tieing a game counted as being one of the last players standing (e.g., the last two players alive were eliminated at the same time). For both sets of tests, every time we conducted a new test where all participating players were placed randomly with replacement in one of 8 evenly spaced locations on the perimeter of the board. The results obtained are shown on a 95% confidence interval using normal distribution for average depth, and Bernoulli distribution for the win rate.

All Implemented Algorithms vs Each Other 4 Players			
Algorithm	**Depth**	**Win %**	**Win % w. Ties**
Max^n	7.21 ± 0.01	(12.78%, 14.12%)	(16.77%, 18.26%)
Max^n P+	12.85 ± 0.03	(17.76%, 19.28%)	(22.19%, 23.84%)
Paranoid	9.36 ± 0.01	(17.37%, 18.88%)	(22.67%, 24.33%)
Paranoid P+	15.58 ± 0.03	(24.82%, 26.53%)	(30.15%, 31.96%)
B.R.S.	7.21 ± 0.01	(18.96%, 20.52%)	(24.56%, 26.27%)
B.R.S. P+	9.63 ± 0.03	(24.09%, 25.78%)	(29.86%, 31.67%)
Multi-Minimax	9.17 ± 0.01	(21.37%, 23.0%)	(27.35%, 29.11%)
Multi-Minimax P+	17.09 ± 0.03	(35.69%, 37.58%)	(42.3%, 44.24%)

Fig. 9: The results of running each one of the implemented algorithms *vs* each other. 20,020 tests were run for each algorithm. In the tables, P+ implies that the scheme has been enhanced with Added Pruning.

All Implemented Algorithms vs Each Other 6 Players			
Algorithm	**Depth**	**Win %**	**Win % w. Ties**
Max^n	7.13 ± 0.01	(8.77%, 9.70%)	(11.12%, 12.14%)
Max^n P+	9.78 ± 0.01	(10.77%, 11.78%)	(13.24%, 14.34%)
Paranoid	8.87 ± 0.01	(12.11%, 13.17%)	(15.16%, 16.33%)
Paranoid P+	11.79 ± 0.03	(15.10%, 16.27%)	(18.55%, 19.81%)
B.R.S.	5.94 ± 0.01	(12.85%, 13.94%)	(16.08%, 17.27%)
B.R.S. P+	6.77 ± 0.02	(14.47%, 15.61%)	(18.18%, 19.43%)
Multi-Minimax	8.53 ± 0.01	(16.12%, 17.31%)	(19.93%, 21.22%)
Multi-Minimax P+	14.42 ± 0.02	(25.75%, 27.16%)	(30.62%, 32.10%)

Fig. 10: The results of running each one of the implemented algorithms *vs* each other. 20,020 tests were run for each algorithm. In the tables, P+ implies that the scheme has been enhanced with Added Pruning.

6 Conclusions

In this paper, we have considered the problem of playing multi-player games, when there is no fixed move ordering between the players. The game that we have used to demonstrate our hypothesis is "Light Bike". We have clearly shown that for the given heuristic used for the game, our newly-proposed scheme, the Multi-Minimax, and the added pruning methods, were more effective than alternative solutions. Both methods provided largely increased search depth along with a minimal loss in exploring important board states, thus being more effective. In the general case, we have not shown that Multi-Minimax is more effective than Paranoid, Best Reply Search and Max^n [1, 4–6]. But, to make sure the algorithms were implemented correctly, they were all placed against randomly moving opponents to show the effectiveness, and to also share common code.

Future work in testing out Multi-Minimax could prove its ineffectiveness in games where players can much more easily work together to eliminate other players due to it being too similar to Minimax. Similarly, Multi-Minimax could be ineffective in games where searching realistic board states is much more important than having improved search depth. Thus, we can assume that at least in the area of fixed sequential turn-based games, Multi-Minimax will not perform well in scenarios where Best Reply Search does not. Having added pruning methods for search-tree based AI algorithms playing other types of turn-based games could also prove to be effective. Experimentation with combining supervised learning, Monte-Carlo tree search, SSS* pruning or pruning for simultaneous move search-trees with the Multi-Minimax algorithm, could also be effective [10–13]. For example, one could train a machine learning algorithm to predict the depth at which Multi-Minimax should start at during iterative deepening.

References

1. Schadd, Maarten PD and Winands, Mark HM: Best reply search for multi-player games. IEEE Transactions on Computational Intelligence and AI in Games **3** (1) (2011) 57–66
2. Knuth, Donald E and Moore, Ronald W: An analysis of alpha-beta pruning. Artificial intelligence **6** (4) (1975) 293–326
3. Sturtevant, Nathan R and Korf, Richard E: On pruning techniques for multi-player games. **49** (2000) 201-207
4. Sturtevant, Nathan: A comparison of algorithms for multi-player games. International Conference on Computers and Games (2002) 108–122
5. C. Luckhardt and K. Irani: An Algorithmic Solution of N-Person Games. Proceedings of the AAAI'86 (1986) 158–162
6. N. Sturtevant: Multi-Player Games: Algorithms and Approaches. University of California (2003)
7. C. E. Shannon: Programming a Computer for Playing Chess. Philosophical Magazine **41** (1950) 256–275
8. Korf, Richard E: Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence **27** (1) (1985) 97–109
9. Campbell, Murray S and Marsland, T. Anthony: A comparison of Minimax tree search algorithms. Artificial Intelligence **20** (4) (1983) 347–367
10. Stockman, George C.: A Minimax algorithm better than alpha-beta?. Artificial Intelligence **12** (2) (1979) 179–196
11. Buro, Michael: Improving heuristic mini-max search by supervised learning. Artificial Intelligence **134** (1-2) (2002) 85–99
12. Saffidine, Abdallah and Finnsson, Hilmar and Buro, Michael: Alpha-beta pruning for games with simultaneous moves. Twenty-Sixth AAAI Conference on Artificial Intelligence (2012)
13. Baier, Hendrik and Winands, Mark HM: Monte-Carlo tree search and Minimax hybrids. 2013 IEEE Conference on Computational Inteligence in Games (CIG) (2013) 1–8
14. Korf, Richard E: Multi-player alpha-beta pruning. Artificial Intelligence **48** (1) (1991) 99–111
15. Sturtevant, Nathan R: Last-Branch and Speculative Pruning Algorithms for Maxˆ n. IJCAI **3** (2003) 669–678
16. Sturtevant, Nathan R: Leaf-value tables for pruning non-zero-sum games. International Joint Conference on Artificial Intelligence **19** (2005) 317