Gesellschaft für Informatik (Hrsg.): SKILL 2019,
Lecture Note in Informatics (LNI) – Seminars, Gesellschaft für Informatik, Bonn 2019 1

Adversarial N-player Search using Locality for the Game of Battlesnake

Maximilian Benedikt Schier[1], Niclas Wüstenbecker[2]

Abstract: This paper presents an approach to designing a planning agent for simultaneous N-player games. We propose to reduce the complexity of such games by limiting the search to players in the locality of the acting agent. For Battlesnake, the game at hand, an iterative deepening search strategy utilizing both alpha-beta and max^n search is suggested. Useful metrics for estimating player advantage are presented, especially using a diamond flood filler for measuring board control. Furthermore, the process of our heuristic parameter tuning with a grid search and a genetic algorithm is described. We provide a qualitative analysis of our algorithm's performance at the international artificial intelligence competition Battlesnake, Victoria. Here, our agent placed second in the intermediate division.

Keywords: Game Tree Search; Artificial Intelligence; N-player Game; max^n; Battlesnake

1 Introduction

Snake is a very popular game, well known from implementations on many arcade machines, personal computers and especially cell phones. The game of Battlesnake is a multiplayer variant of this classic where many snakes simultaneously fight for food and survival. The eponymous Battlesnake competition in Victoria, B.C., Canada is an annual event focusing on the development of artificial intelligence. Each team designs and implements an agent to slither their snake to victory. Competitors are grouped in multiple division depending on skill, with the intermediate division primarily targeting university students interested in the field of AI.

Traditionally, many snakes are based on simple reflex agents [RN03]. More cunning teams employ a game tree search to plan ahead. An especially popular choice in this domain is alpha-beta search because of its simplicity, efficiency, and familiarity. However, it is not a viable solution for Battlesnake as it is limited to two player games and therefore can not be used effectively. We apply a lesser-known variant of minimax, namely max^n, and describe our experience, the process of our development and the parameter tuning of the heuristic using a genetic algorithm.

[1] Leibniz University Hannover, Institute for Information Processing, Appelstr. 9a, 30167 Hannover, Germany
schier@tnt.uni-hannover.de
[2] Leibniz University Hannover, Institute for Information Processing, Appelstr. 9a, 30167 Hannover, Germany
wuestenb@tnt.uni-hannover.de

Our main **contributions** are:

- We propose a general iterative deepening tree search approach using alpha-beta and max^n search which reduces tree complexity using locality for N-player games
- We define a measure of locality for the game of Battlesnake, and adapt our algorithm
- We introduce several useful metrics to estimate the advantage of leaf nodes during search for Battlesnake
- Our heuristic parameter tuning approach using genetic optimization is outlined
- We perform a qualitative analysis of our agent in the context of the international artificial intelligence competition Battlesnake, Victoria
- Our implementation of the agent as described in this paper, which is also the version used during competition, is made publicly available[3]

2 Related Work

To the best of our knowledge, no prior work has been published on the game of Battlesnake. However, some work was published about search complexity reduction for general N-player games with max^n search. Lisý *et al.* [Li10] suggest splitting a game into multiple potentially overlapping subsets of agents using game locality. These subsets may then be searched independently and are ultimately combined to form a joint plan. This approach shows many similarities to ours with regard to agent selection and masking but the play out and combination of all subsets render it more complicated to implement.

Another optimization to max^n proposed by Lisý *et al.* is a procedural knowledge heuristic to guide tree search towards each agent's goal in their goal-based game-tree search algorithm [Li09]. However, this work uses hand-crafted or inferred opponent models, which have to be implemented on a per-game basis. This is generally much harder than simply defining the locality of agents. Whereas our method might reduce the accuracy of search by masking agents, this method does so by cutting branches which are considered unlikely to be selected.

A completely different approach to solving games with a discrete action space which has gained popularity has been reinforcement learning, for example with Deep Q-networks (DQNs). These networks learn to make intuitive decisions and have the potential to exceed all traditional heuristic-based agents as they can find metrics that are too complicated to be hand-engineered or simply not obvious to humans. Monte Carlo tree search using DQN value heuristics is already outperforming alpha-beta tree pruning for a variety of two-player games like chess and GO [Si18]. However, DQNs still require a lot of resources and time to be trained and their decision making is not transparent.

[3] https://github.com/m-schier/battlesnake-2019

3 Battlesnake

The game Battlesnake is a multi-agent variant of the popular game Snake. In the classic game, a single player controls a snake consisting of multiple tiles on a square grid. Each turn the player can move the snake to an adjacent tile, afterward, the tail disappears. If a snake consumes randomly spawned food, it grows a tile as the tail remains during that turn. The same game mechanics apply to Battlesnake, however, the game adds more complexity. Consumed food will be placed on a free tile again with an exponentially increasing probability per turn. Failing to consume any food within 100 turns causes death by starvation. Additionally, a collision of the snake head with any other body part or trying to move outside of the board boundaries (wall collision) also kills the colliding snake immediately. If two or more snake heads move into the same tile simultaneously, all but the largest snake die, unless there is no difference in length between the largest, in which case all snakes die resulting in a draw.

Two to eight snakes play on one game board simultaneously with predetermined starting positions. The game board is always square and has one of the following sizes: 7×7, 11×11 and 19×19. An example board configuration is presented in figure 1.

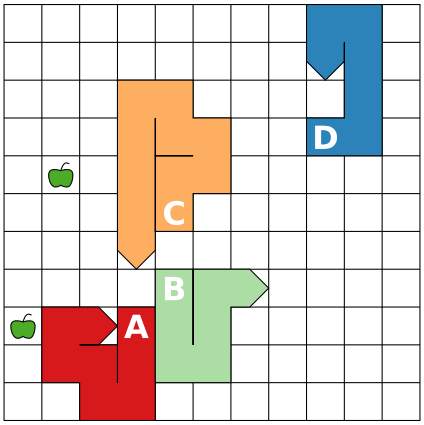


Fig. 1: Board state taken from one of our tournament games. Snake heads are marked with capital letters, tails are triangularly shaped. Food is depicted as an apple. In this state, *A* and *C* may cooperate to kill *B* in the next frame if *A* moves up and *C* down, leaving *B* with no option but running into itself or into the head of one of the longer snakes, which is exactly how this game played out.

4 Agent

An agent playing a game of Battlesnake is presented the current game state each frame and selects an appropriate movement action. In order to find a suiting algorithm for our agent, the game's characteristics were first classified:

Discrete There are always four actions a player may choose (North, East, South, West).

Not deterministic Food spawn positions are not known to agents and determined by a pseudo-random number generator.

N-player The game (usually) has more than 2 players.

Simultaneous In each state each agent must choose an action.

Fully observable The entire board as well as the attributes of each player are observable for every player.

Zero-sum game The outcome of a game of Battlesnake is only determined by the ranking of the players when the second to last player dies (the final state), where the ranking is determined by the frame in which a snake died, later being better. Consequently, each ranking position can be assigned a score such that the sum of all scores is always zero. If two or more snakes die in the same frame, this is considered a tie, in such a case the score of all ranking positions the snakes finished on will be averaged to still reach a sum of zero.

4.1 Tree Search

After classification, we evaluated that an agent using a tree search is well suited due to the game's small discrete action space, fully observable state and zero-sum nature. However, its non-deterministic, non-sequential play out is problematic for search. On the largest board size of 19×19 at the start of the game, the number of empty tiles where food may spawn is larger than 300 as few tiles are occupied. This spawn chance leads to a very high branching factor, which would limit search depth too severely to even warrant a tree search. Therefore, it was decided to not spawn any new food during game tree play-out making the game deterministic. The lack of food spawn would have to be compensated by some heuristic that measures how many tiles on the board are reachable for each snake.

Secondly, the simultaneous moves are of concern as simultaneous play requires forming a strategy to select the best action using a game equilibrium in unstable states where one action is not clearly dominant. We employ the concept of delayed move execution to sequentialize the search, where instead of performing all actions at once as usual, we apply each agent's action in sequence and only update the state with the final agent's move. During search, we follow the trivial pure strategy of always selecting the action with best minimax value. Such a strategy is not always stable and playing against a better, mixed strategy may not be optimal either, but does allow the game to be viewed in a purely sequential tree.

With these simplifications, Battlesnake is a discrete deterministic sequential multi or two-player zero-sum game. For games with more than two players, the $\max^n$ search algorithm described by Luckhart and Irani [LI86] is used, which is a tree search algorithm where each player executes their move in turn and always selects the move giving them the best payoff for the given state. When only two players are left, a switch to alpha-beta pruning [KM75]

is performed. Alpha-beta pruning reduces the number of examined nodes in a minmax tree search by not playing out subtrees that an optimally playing opponent would not select.

4.1.1 Iterative deepening with adaptive play out selection

Iterative deepening search is often employed to find the best available action with a given time constraint and has been widely applied to alpha-beta and depth first search. While iterative deepening can be used just as easily with $\max^n$ search, further optimization can be performed for any game with a notion of locality. Complete play out of all agents is not required if an opponent has minimal influence on our decision making. Any agent that may not be directly interacted with within the search depth, usually because of spatial locality, may be ignored. Choosing to not play out (mask) agents can greatly reduce complexity in these scenarios. In case of masking all but two agents, instead of $\max^n$ search much more efficient alpha-beta pruning may even be used. We propose the general *IDAPOS* algorithm outlined in algorithm 1 for adaptive play out selection in N-player games. While the elapsed time t_{Elapsed} is less than the specified cutoff time t_{Cutoff}, the played out players p_{played} for the given depth and state are selected using a game specific function. The current state s is then masked to the state s_{masked} only containing played out players using a game specific masking strategy. Depending on number of played out players, the appropriate search algorithm is used and the best action a_{best} updated. Search depth is increased per loop iteration.

Algorithm 1 General iterative deepening game tree search with adaptive play out selection

```
function IDAPOS(s, t_Cutoff)
    depth ← 1
    while t_Elapsed < t_Cutoff do
        p_played ← SelectPlayedOut(s, depth)
        s_masked ← Mask(s, p_played)
        if |p_played| = 2 then
            a_best ← AlphaBetaSearch(s_masked, depth)
        else
            a_best ← MaxNSearch(s_masked, depth)
        depth ← depth + 1
    return a_best
```

This algorithm can be adapted to any game by implementing the selection and masking functions. The selection returns the set of players that may directly be interacted with within the cutoff depth, the masking hides all players not selected for search. We propose the following masking strategies for the general case, which strategy to use should be decided on a per-game basis:

Removal All masked agents are removed from the board.

Freezing All masked agents remain in their current state even if they have to perform an action.

Simple move The masked agents perform basic actions that require minimal computation time while keeping them alive.

4.1.2 Adaption for Battlesnake

Play out selection Due to the spatial locality of Battlesnake, situations were players are too far apart to interact directly within the search depth cutoff are often given. In these scenarios such players may be masked. Especially in the opening stages of the game, where up to eight players are present, planning an effective local strategy against neighboring players is much more important than correctly predicting developments at the far side of the board. The theoretical reduction of nodes evaluated compared to standard $\max^n$ is shown in Figure 2. A significant improvement can be observed, especially on large board sizes. It should be noted that due to the symmetry of the board in starting positions, no alpha-beta pruning was used by *IDAPOS* in this figure, as it is impossible to be close to exactly one opponent.

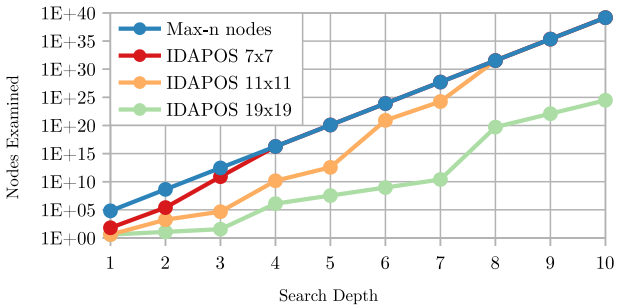


Fig. 2: Number of examined nodes with increasing search depth using *IDAPOS* presented in algorithm 1 with selection algorithm 2 for different board sizes in comparison with total tree playout. Snakes are at their starting positions.

For play out selection, all agents are chosen which the player could possibly interact with within the current search cutoff, where an interaction is either a head-on-head or head-on-body collision. Since the head of a snake moves into its 4-neighborhood each frame, the Manhattan or L_1 distance is an accurate lower bound for the number of turns required to reach a given tile. Thus any snake with its head within an L_1 distance of twice the search depth or less may be interacted with through head-on-head collision and with any body part within an L_1 distance of exactly the search depth or less may be interacted with through head-on-body collision. Algorithm 2 outlines the selection process.

Masking Removing masked agents from the board is the worst option for this game. Doing so opens up spaces on the board that should not be accessible, i.e. dead ends. In later

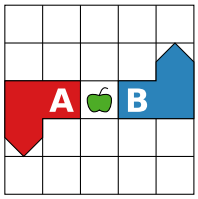
Algorithm 2 Play out selection for the game of Battlesnake

```
function SelectPlayedOut(s, depth)
    p_played ← ∅
    for all p ∈ s.players do
        if Manhattan(p.head, s.self.head) ≤ 2 · depth then
            p_played ← p_played ∪ {p}
        else
            for all ψ ∈ p.parts do
                if Manhattan(ψ, s.self.head) ≤ depth then
                    p_played ← p_played ∪ {p}
                    break
    return p_played
```

stages of the game, snakes occupy dozens of tiles, freeing these up by removal changes the situation on the board greatly. Freezing is the safest option as there is no risk of suicidal behavior compared to a very basic simple move like running into a dead end. Making a simple move is the most realistic, but also the most complex due to time constrains. We developed a simple move approach where the agent tries to maintain direction while dodging collisions with itself and snakes of equal or larger length.

Locally connected groups An agent A may be too far from our own agent to affect our decision making but close enough to influence an agent B we are playing out. Thus our simulation of B's behavior will be flawed as it does not consider agent A acting intelligently. To fix this issue, A would have to be included in the tree search, which is not desirable as A might invoke the same issue on a third agent, requiring the simulation of all locally connected agents. In order to keep the branching factor and tree complexity low, inaccuracies caused by this problem are tolerated.

Distortion of the heuristic by masking Another issue arises when not including the masked agents in the advantage calculation. Given the situation in figure 3 it is desirable for both agents to commit to eating, resulting in a tie. However, if the shown tiles were only part of the board with a third agent present out of view, it would not be in the best interest of both agents to commit to eating the food as the third agent would win. To resolve this issue while preserving the zero-sum nature of the game, an additional punishment for the own agent death was added. This offsets any disadvantage through weaker positioning or growth as predicted by our heuristic if the agent dies in a tie. Consequently, our agent only goes for a tie if the other options all lead to a loss. An additional implication of this modification is that all opponents are rewarded slightly for cooperating against our agent during the search, which is a strategy they are unlikely to actually follow.



A payoff		B moves			
		West	North	East	South
A moves	West	**-1**	**-1**	**0**	**-1**
	North	-0.5	0	**1**	0
	East	**0**	0.5	**1**	0.5
	South	-0.5	0	**1**	0

Fig. 3: A typical tie situation encountered in Battlesnake. Payoffs for A are given with terminal states marked bold. The heuristic is assumed to reward greater length by eating with 0.5 points advantage and being the sole survivor with 1 point advantage. Going for the food dominates all other actions for both A and B. However, carrying out their best available option kills them both in a head on head collision as they have equal length.

4.1.3 Scoring Leaf Nodes

Some additional biases were introduced in the scoring of leaf nodes to improve the agent's behavior. In general, it may be beneficial to draw out a game as long as possible when facing defeat. Because we used a compiled language and had optimized our search implementation using profiling, we were confident that we could beat many competing agents in terms of search depth. Therefore, we rewarded surviving as long as possible given the chance the opponent was unaware of a branch due to his more restricted search depth. Another bias very specific to this game that was implemented for leaves was scoring of death reasons. It is preferable to die in an uncertain event rather than in a certain one. For example, given the choice between starving, and eating food but risking a head-on-head collision, it is always preferable to eat, as death is not certain. Alternatively, when the agent may only choose between colliding with a wall, its own body, or an enemy body, it should always go for the enemy. The enemy may die in the same frame but the wall or its own body won't disappear. Therefore, the death reasons were biased in the following order from most favorable to least: head-on-head collision, head-on-body collision, starvation, own-body collision, wall collision.

4.2 Metrics

After analyzing a promising search strategy, the next step is to evaluate a game position on the board by applying a heuristic, consisting of several metrics. The computation time of a metric is very limited, as a longer computation time drastically decreases the search depth of our tree search.

Length Advantage The first metric we propose is a snake's advantage in length. For the normal game of Snake, this is equivalent to the score, however, one has to be careful as a longer snake is also less versatile and can accidentally box itself in more easily. In general,

however, we found that length is an advantage because the snake body acts like an obstacle enemies have to avoid. To compute this metric we subtract the overall average snake length from our own length.

Board Control In addition to the length advantage, it is very important to be able to reach a tile faster than the enemy, or in other words: controlling a tile. This concept of tile control can be seen in a large number of board games, most notably the game of chess [Ha75]. In the past, many approaches used a Voronoi diagram [AK00] to estimate the number of controlled tiles. However, this method is not very accurate and can not be expanded easily. Therefore, we decided to count the number of controlled tiles using an efficient implementation of a queue-based diamond flooder [Le81] in adversarial operation. The flood fill algorithm starts with the queue initialized with each snake's head position, sorted by snake length in descending order. Since a snake's head moves into one of the directly adjacent four tiles each turn, a diamond flooder is used. The queue ensures that the snakes take turns filling out their next available tiles. The initial order guarantees that the largest snakes get to start each fill step. Thus contested tiles that multiple snakes may reach at the same time are attributed to the largest one. This fill method returns the exact number of tiles a snake can reach before any enemy. An example of a flood fill visualization can be seen in figure 4. Here, snake *B* has better control over the empty tiles and can, therefore, reach more potential food spawn locations first.

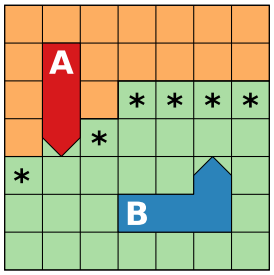


Fig. 4: Adversarial flood fill performed starting from the head of both snakes. Empty tiles are colored matching the snake they are controlled by. Tiles marked $*$ are contested and could be reached by both snakes in the same amount of turns but are attributed to the larger snake *B*. *B* could force reaching the tile if both players go for it. The smaller snake *A* would die in a head-to-head collision.

Snake Health Another metric is the snake's health. As described in section 3 every snake can make 100 moves without eating food before starving. Therefore it is crucial to include this in the heuristic. A simple linear mapping of the health to a value in range $[0, 1]$ would not be sufficient, because this would favor eating food far too much. We factored in the distance to the nearest food computed by using the flood fill described above. The number of turns until starvation should always be greater than the distance to the nearest food item.

5 Experiments and Results

Based on the presented metrics, a heuristic can be constructed by weighting the metrics described in section 4.2. We first started by hand tuning the coefficients used to weight each metric. However, such an approach is not suited for smaller improvements as there is no systematic way of assessing the heuristic. Since an agent's performance using a fixed search depth solely depends on the heuristic used, the search depth for all agents was reduced and a game could be simulated much faster. By playing out a large number of games we were able to calculate the average leaderboard position and thus gather statistical evidence on whether an agent's strategy outperforms the others. Due to the game's adversarial nature, it is not certain that self-play alone can result in the best strategy as there may be a counter-strategy that exploits it. To prevent this, multiple strategies from last years tournament were implemented: a greedy strategy that tries to consume food as fast as possible, an aggressive strategy which builds on the greedy strategy but attacks another snake when it is longer, and a tail chasing strategy that minimizes the distance between head and tail of the snake until the health is very low and then acts greedy until food is consumed. Any heuristic change could be validated by testing it against the simple agents and against the original heuristic before the change. To roughly estimate the importance of a metric a simple grid search was applied.

Genetic Optimization and Tests To further optimize the weights, a genetic algorithm which adjusted all coefficients of the heuristic using a bit string mutation on a string of 6 bits for each metric was used. The bit string represented a number that was mapped to the range determined by the grid search. We used the average leaderboard position as a fitness measure for the population. A total of 10,000 games with low search depth were carried out. The decisive advantage the heuristic found through genetic optimization has over other heuristics tested is shown in Table 1. The table on the right shows that our *IDAPOS* algorithm performs better than other tree search algorithms when using the same heuristic and same search time limit. While standard deviations are high due to the non-deterministic nature of the game, both our algorithm as well as heuristic are found to be statistically significantly better than the next best one using t-tests with $\alpha = 0.01$.

Heuristic	Average position
Genetic Optimization	3.33 (±2.21)
Greedy eater	4.15 (±2.31)
Aggressive	5.02 (±2.13)
Tail chaser	5.51 (±1.87)

Algorithm	Average position
IDAPOS	4.12 (±2.13)
$\max^n$	4.51 (±2.21)
alpha-beta	4.64 (±2.35)
MinMax	4.72 (±2.41)

Tab. 1: Average leaderboard positions after $n = 1,500$ simulated playouts with an eight player board and search time limit of $100ms$. Left table shows different heuristics with alpha-beta search, while right table shows different algorithms with our heuristic found through genetic optimization. Standard deviation is shown in brackets.

Tournament Results Our team *Niedersächsische Kreuzotter* placed second in the intermediate division which consisted of 65 teams. It became apparent during the tournament that our agent tried to optimize the number of controlled tiles very well but sometimes failed to convert its length advantage into a real advantage. In other cases the agent focused too much on board control, neglecting to eat food placed at the edge of the board. This can be attributed to the fact that the agent would have to give up board control in order to eat. In these cases, the agent did not reach sufficient length and could easily be pressured by longer opponents. For these reasons, we, unfortunately, lost in the final stand-off. Overall, however, our agent played especially well during the opening stages of the game when most snakes were still alive. We mainly ascribe this to our $\max^n$ search approach. The agent performed extraordinarily well in an event sponsor's unofficial exhibition tournament where each agent faced an increasingly large number of cooperating hostile agents, strongly outperforming participants of all divisions and winning a special price.

6 Conclusion and Outlook

In this paper, we introduced the game of Battlesnake as well as the international artificial intelligence competition of the same name. The game was classified and an iterative deepening tree search algorithm using both alpha-beta pruning as well as $\max^n$ search was implemented. Our algorithm reduces the number of played out agents using the locality of the game by masking players which may not directly interact with ourselves within the depth limit. Special considerations required to perform a tree search with masked players in general, as well as specifically for this game, were outlined. We suggested several metrics which are useful for assessing non-terminal leaves during the search, notably, establishing a flood fill based metric for evaluating board control. Genetic optimization was used to fine tune coefficients and balance the influence of different metrics on the overall heuristic, outperforming both grid search as well as our initial hand-crafted heuristic. Our proposed *IDAPOS* algorithm was found to outperform both $\max^n$ and alpha-beta search in this game using the same heuristic. Finally, a qualitative analysis of our approach compared to other tournament competitors showed that our agent generally outperformed the competition when many players are present on the board.

In the future, the efficiency of N-player search could be improved. As $\max^n$ employs no pruning, it can generally only search a more shallow tree than alpha-beta pruning with a similar branching factor. In most of our game situations, the $\max^n$ search depth was only four moves ahead while alpha-beta search computed up to eight moves ahead. Recent improvements in pruning N-person game trees have been made, notably, HyperMax [Fr14]. As outlined in our approach, the agent currently follows the pure strategy of selecting the action with the best minimax value. However, situations such as shown in figure 3 are unstable if more than two players are present. In these unstable situations, an accurate opponent model would increase the agent's expected payoff. Sturtevant *et al.* propose the probabilistic $\max^n$ tree search prob-$\max^n$. This can outperform classic $\max^n$ search in expected pay-off despite reducing the total number of nodes evaluated [SZB06].

7 Acknowledgements

We would like to thank the Institute for Information Processing at our university, the Leibniz University Hannover. Without the support of the Institute, especially Prof. Bodo Rosenhahn, we would not have been able to attend the competition in Victoria. Additionally, thanks to Florian Kluger and Frederik Schubert who provided guidance and accompanied us on the trip to Canada. More over, we would like to thank Christoph Reinders for his advise on this paper. Lastly, we want to thank all organizers and sponsors of the Battlesnake competition for making this event possible.

References

[AK00] Aurenhammer, Franz; Klein, Rolf: Voronoi diagrams. Handbook of computational geometry, 5(10):201–290, 2000.

[Fr14] Fridenfalk, Mikael: N-Person Minimax and Alpha-Beta Pruning. In: NICOGRAPH International 2014, Visby, Sweden, May 2014. pp. 43–52, 2014.

[Ha75] Harris, Larry R: The heuristic search and the game of chess a study of quiescence, sacrifices, and plan oriented play. In: Proceedings of the 4th international joint conference on Artificial intelligence-Volume 1. Morgan Kaufmann Publishers Inc., pp. 334–339, 1975.

[KM75] Knuth, Donald E; Moore, Ronald W: An analysis of alpha-beta pruning. Artificial intelligence, 6(4):293–326, 1975.

[Le81] Levoy, Mark: Area flooding algorithms. Two-Dimensional Computer Animation, Course Notes 9 for SIGGRAPH, 82, 1981.

[LI86] Luckhart, Carol; Irani, Keki B: An Algorithmic Solution of N-Person Games. In: AAAI. volume 86, pp. 158–162, 1986.

[Li09] Lisỳ, Viliam; Bošanskỳ, Branislav; Jakob, Michal; Pěchouček, Michal: Adversarial search with procedural knowledge heuristic. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2. International Foundation for Autonomous Agents and Multiagent Systems, pp. 899–906, 2009.

[Li10] Lisỳ, Viliam; Bošanskỳ, Branislav; Vaculín, Roman; Pěchouček, Michal: Agent subset adversarial search for complex non-cooperative domains. In: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games. IEEE, pp. 211–218, 2010.

[RN03] Russell, Stuart J.; Norvig, Peter: Artificial intelligence - a modern approach, 2nd Edition. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.

[Si18] Silver, David; Hubert, Thomas; Schrittwieser, Julian; Antonoglou, Ioannis; Lai, Matthew; Guez, Arthur; Lanctot, Marc; Sifre, Laurent; Kumaran, Dharshan; Graepel, Thore et al.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419):1140–1144, 2018.

[SZB06] Sturtevant, Nathan; Zinkevich, Martin; Bowling, Michael: Prob-Maxˆ n: Playing N-Player Games with Opponent Models. In: AAAI. volume 6, pp. 1057–1063, 2006.