Mastering Strategies in a Board Game of Imperfect Information for Different Search Techniques

Michał Przybylski
Faculty of Computer Science and Management
Wrocław University of Science and Technology, Poland

Dariusz Król
Department of Information Systems
Wrocław University of Science and Technology, Poland
Email: Dariusz.Krol@pwr.edu.pl

***Abstract*—To the best authors' knowledge this work is the first to develop a full computer implementation of *The Great Turtle Race* (GTR), a complex board game characterized by several uncertainties that uses computational techniques to evaluate board positions and select the best move. In the game, a novel combination of popular propagation-based optimization techniques and four playing strategies is implemented. One of the main goals of this study is to determine how to generate opponents that are quick and safe to play against, rather than being necessarily superior. The paper starts by a brief overview of the game and its rules, followed by some analytical results that emerge from its characteristics. It then moves to provide relevant reinforcement learning methods by which Monte Carlo tree search, minimax and alpha-beta pruning were implemented. The validity of the concept is finalized by a series of experiments, in which these algorithms and strategies were successfully verified against each other.**

***Keywords*—Monte Carlo tree search, minimax search, alpha-beta pruning, zero-sum game, reinforcement learning.**

I. Introduction

Much of the work on artificial intelligence and games was directed toward classic board games, such as tic-tac-toe, chess, and checkers [1]. Board games can now, in most cases, be played by a computer at a higher level than the best humans. Games usually implement brute force methods of search to produce the very best players. There are many games still waiting to be explored from this perspective. *The Great Turtle Race* (*Schildkrtenrennen* in German) designed by Reiner Knizia and publish in 2004 is an example of a game with multiple opponents of imperfect information by the fact that some of the information is hidden from all of the players [2]–[3]. Unlike complete information games in which the techniques to solve the game have been known and understood for a long time, games of imperfect information have not received the same sort of interest [4]–[6]. In our opinion this research yields first full computer implementation with integrated various intelligent capabilities to this game.

There are several issues to consider when evolving a game strategy: (a) fitness function, (b) exploration, (c) implementation, (d) game interface, (e) input representation and agent architecture, (f) play depth and (g) reinforcement learning method [7]–[8]. In this paper, we aim to investigate the merits of applying learning methods to evaluate board positions and to select the best move. We explore three popular techniques: Monte Carlo tree search (MCTS), minimax, alpha-beta, and adopt four strategic decision-making approaches: the fast (F) strategy, the efficient-smart (E) strategy, the hybrid (H) strategy and the dummy (D) strategy with the legal moves choose at random [9]–[10].

978-1-5090-5795-5/17/$31.00 © 2017 IEEE

The paper is structured as follows. First, we give the necessary technical background for Turtle Race game before introducing win strategies and explaining them in detail. Next three optimization approaches to solving the defined strategies were compared. We then take an empirical approach to show their effectiveness. We conclude the paper pointing out some ideas for future work.

II. Modelling the GTR Game

A. Game Rules

The goal of this game is to move turtles from the start square to the tenth square. In a single game can participate from two to five players and all turtles (red, green, blue, yellow and purple), regardless of the number of players. Afterwards, each player draws a tile, which represents the colour of a turtle. They have to hold it back. The rest of tiles are covered during the game. Next the pack of cards is shuffled and each player draws five cards. Players should hold cards in such a way that no one else can know them. The remaining cards are placed face down, forming the draw pack.

The cards are used for deciding how the turtles will be moved. Players take turns until at least one turtle reaches the goal. Moves are fully dependent on active player. He decides, which card from his hand he uses. A card should be placed face up to all players. After that the action symbolized on that card is performed, then the card is placed on the discard pile. Each card represents the following turtle movements:

- *double plus* [++] means two squares forward,
- *plus* [+] means one square forward,
- *minus* [−] means one square backwards, this card can not be used on the start square,
- *double arrow* [⇑⇑] means two squares forward but only to the turtles are the nearest to the start square. If on this position is more than one turtle, a player selects which one he wants to move,
- *arrow* [⇑] means one square forward cognately to *double arrow*.

A player moves a turtle printed on played card but turtles can be stacked. When moving one turtle, all turtles that are

Fig. 1: A predefined mid-game position.

Fig. 2: Red turtle moves one square forward [⇑].

Fig. 3: Purple turtle moves one square backwards [−].

stacked on its back also move at the same time. Turtles that are stacked below do not move. When a turtle reaches a square already occupied by one or more stacked turtles, it is placed on top of that stack. If a card symbolizes multicoloured turtle, player chooses, which turtle is moved. The game ends when first turtle reaches the food, it means reaches the last square. Then, each player reveals a self-drawn tile. The player of the last square turtle is the winner. When there are more turtles on the last square, the winner is the player of the lowest one in the stack.

B. Example of Game Play Scenario

To give an example of game play (Fig. 1), we let the target player to draw the multicoloured turtle card with an *arrow* [⇑] and then move the red turtle one square forward. The yellow turtle, which is on the red back, also moves one square forward. In contrast to the yellow one the blue turtle remains unchanged because it is stacked below the red one (Fig. 2). Next player draws a card with purple turtle with minus [−]. The purple one has to move one square backwards with turtles: yellow and red, which are stacked on the purple one (Fig. 3). Another player takes a red turtle card with two pluses [++] and moves the red one two squares forward. Of course, the yellow one moves forward too (Fig. 4).

C. Formal Complexity Description

In order to define a game complexity, it is necessary to introduce the evaluation measures of finding games solutions. In this context two ways of measuring game complexity occur in the majority of games [11]:

Fig. 4: Red turtle moves two squares forward [++].

- *State-space complexity* which is the number of legal game positions reachable from the initial position of the game.
- *Game-tree complexity* which is the number of leaf nodes in the smallest full-width decision tree that establishes the value of the initial position.

The above-mentioned measures, despite their generality, remain one of most popular ways to assess the complexity of games [12]. Having a certain understanding of a game complexity, we can move to the analysis. First, we should familiarize with the possibilities and limitations of the analysed game. *The Great Turtle Race* has a determined set of moves, each one described in terms of conditions of its performance and changes of state in the game. The goal of the game is also clearly defined - it is one and identical for each player. Thereupon, independently of the applied strategy it is to gain one concrete, immutable result [2]. This state in the game guarantees the successful termination for only one player and excludes the chance of collective win. The following list summarizes the notations used in this paper:

- S denotes a set of available positions,
- C denotes a set of available colours: red, green, blue, yellow, and purple,
- P denotes a number of players: 2, 3, 4, 5,
- M denotes a number of available movements: 3,
- K denotes a number of cards hold by each player: 5,
- F denotes a set of available squares, where $|F| = 10$.

1) State-space Complexity: Before we estimate the size of the state space ($|S|$), it is necessary to look at what constitutes this value. First, it is necessary to consider a number of available squares on board ($|F|$). According to the game rules on each square from one to five turtles can be placed, creating a stack. As an example, if five turtles stay on a square $f \in F$ and they create a stack of colours from C, ordered from down to up in the following sequence: red, purple, blue, green, yellow, then a movement of turtle in colour blue causes a movement of green and yellow ones. This significantly increases a number of available moves, which are calculated respectively as follows:

$$|S| = |C|! * |F| \tag{1}$$

Furthermore, the player has possibility to move selected turtle one square forward or backwards and two squares forward. However not always all movements are allowed. Thus (1) has to be modified to the following form:

$$|S| = |C|! * ((|F| - 2) * M + M - 1) \tag{2}$$

where the number of available movements for the last square equals 0 and for the start square is $M-1$). Substituting variables from (2) leads to: $|S| = 5!*((10-2)*3+2-1) = 3000$.

2) Complexity of Game-tree Search: To correctly consider the computational complexity of game-tree search it is necessary to draw attention to its size. A major aspect of a game is to determine the maximum movement capacity necessary to finish game. As a consequence, it leads to define the maximum size of a game tree, and thereupon it is necessary to prove, that there is always a solution.

But now suppose that we consider a case with $P = 2$ and first player draws red colour, and the second blue colour. We assume, that red player will pursue his own individual interest and blue player will handicap that pursuing. It is expected that any possible combination of cards, which can influence on red turtles position are: [⇑⇑] colourful ($\times 2$), [⇑] colourful ($\times 3$), [++] red ($\times 1$), [+] colourful ($\times 5$), [+] red ($\times 5$), [−] colourful ($\times 2$), [−] red ($\times 2$). Dividing this set on two subsets in a way, that one has only forward moving cards and the other one only moving backwards, we can consider a minimal difference of red turtle movement for one pack shuffling. In such a case, a cardinality for forward movements equals $|RED_+| = 16$, and for backwards $|RED_-| = 4$ respectively.

The essence of this analysis is to investigate the worst case scenario. The blue player has the most unfavourable for the red player sequence: two cards moving backwards the red turtle, two cards moving backwards any colour and one card moving forward the blue turtle. The red player has no cards from RED_+, one card moving backwards the blue turtle and four other cards, not included in RED_+ and RED_-. While the red player does not draw a card moving his turtle forward, cards moving other turtles will be played by both players, including blue one, which belongs to the opponent. In the worst case, cards from RED_+ will be located at the end of the draw pack. Assuming, that all turtles after foregoing moves are located in starting position (forward movements equals backwards movements) and all foregoing cards are used, except from RED_+ and RED_-, then we consider the worst scenario for the red player. In the first four moves the red player will move his turtle one square forward with four cards [+] red. The blue player, in his turn, will deny those movements, moving the turtle one square backwards with cards from RED_-. In addition, we assume all colourful cards [+] were used for moving other turtles but the red one. In the result the turtle stays in place at starting position, all unfavourable cards for the red turtle will be used and remain only cards moving it forward. Describing this state by red_+, which is a subset of RED_+), we can calculate cardinality: $|red_+| = 9$. Given that we are coming to the situation, where each next move causes moving the red turtle forward. This conclusion confirms the thesis that for each pack shuffle finally chosen turtle will change its location towards the finish square, what guarantees to end the game in finite number of moves and provides thereby the finite size of a game tree.

III. Game Implementation: Strategy Selection

Game is often described as any conflictive situation and a player as any participant, which using specified strategy, aims to reach certain goals [13]. Each strategy defines a value, which each player gets, measured in so called usefulness units, which is the measure of players profit. The analysis of this approach allows define a way to represent each state of analysed game:

Definition 1: Let each square has numerical value, such that $p_0 = 0$, $p_1 = 1$,..., $p_9 = 9$. Number of victory points is assigned to players according to the formula:

$$VP(x) = p(x) * 5 + 4 - u(x) \tag{3}$$

where x - selected player, $p(x)$ - square p where is located, $u(x)$ - number of turtles in the stack under selected player.

According to above, each player has designated value of victory points in any moment of game play. It can happen, that players will have same sum of victory points, however that situation is not so important. These values give rudiments of mathematical coverage for delivered strategies, which will be discussed later. Before that, it is necessary to consider four basic movements encountered in a game:

- a forward movement of the own turtle,
- a backward movement of the own turtle,
- a forward movement of the other turtle,
- a backward movement of the other turtle.

To achieve a goal a player adopts a number of strategies. These strategies can be steady-state, dynamic or transient. We analyse and implement the following logically chosen strategies: fast, efficient-smart, hybrid and dummy. The last strategy is the routine in which the move is selected at random during the entire game. Although a dummy strategy has little usefulness for winning games, it can be used as a benchmark to compare performance of other strategies. Due to page limit we only briefly describe the first three strategies.

A. The Fast Strategy

The fast strategy solely relies on minimising the time to reach the last square in the least number of steps. It has significant meaning, because reaching the least number of moves guarantees that players turtle will be the first or one of the firsts finishing the game. Larger value receives that move, which gets player closer to the finish square. And in consequence, the most frequently chosen cards will move own turtle forward, and the least frequently moves own turtle backwards. In order to determine correctly a value of given movement it is necessary to define functions of benefit and efficiency:

- *Position estimation function* will allow assess how important is occupied position depending on type of movement,
- *Movement estimation function* will help unambiguously estimate the weight of available types of movement.

1) Function of position estimation: During the game play turtles occupy next squares separately or in the stack. Each move gets players closer to finish the game and determine the winner. Occupied squares have a significant meaning in strategy developing. The closer the last square is located, the better chances the player has to win. It should be noted that the squares get increasing values. With these preparations and based on definition 1, we obtain the following formula:

$$f_p(i) = p_i \tag{4}$$

where i means the number of a square on which given turtle stands and p_i is the function value of that square.

2) Function of movement estimation: During the course of the game, the state changes. Each attempt brought the players closer or much further to victory. In a single move player has a possibility to move from one do five turtles in two directions: backwards or forward. It depends from their configuration and players cards. For this purpose we introduce the measure of movement given by the following formula:

$$f_d(C) = k(c_p, c_k, c_s) * c_c \tag{5}$$

where

- $C = \{c_p, c_k, c_s, c_c\}$ is a set attributes of the played card and the player itself, where:
  - c_p means colour of turtle belonging to the player,
  - c_k means colour of turtle belonging to the played card,
  - c_s means turtle movement type of the played card,
  - c_c means number of passed squares of the played card,
- $k(c_p, c_k, c_s)$ is the robust rate of movement.

The above-mentioned rate takes values dependent of given parameters according the following formula:

$$k(c_p, c_k, c_s) = \begin{cases} c_c * 10 & \text{when } c_p = c_k \text{ and } c_s \neq \text{" } - \text{"} \\ -c_c * 10 & \text{when } c_p = c_k \text{ and } c_s = \text{" } - \text{"} \\ -c_c & \text{when } c_p \neq c_k \text{ and } c_s \neq \text{" } - \text{"} \\ c_c & \text{when } c_p \neq c_k \text{ and } c_s = \text{" } - \text{"} \end{cases} \tag{6}$$

The combination of those functions leads to the general formula and the final value returned by using this strategy. The formula looks as follows:

$$F_q(i, C) = f_p(i) + f_d(C) \tag{7}$$

The function (9) returns a value of given move using selected strategy. Based on received results, the highest score is selected, and that move is made.

B. The Efficient-smart Strategy

The efficient-smart strategy differs from the previous one. More important are movements, which go against other players. In that way we gain better chances, that opponents will have less chances for putting our turtle behind. In this strategy, we also are able to reach the final square earlier than other players. More valuable will be a movement, which moves opponent further and our turtle closer. This strategy like the previous one defines two functions: *position estimation function* and *movement estimation function*. The only real difference, is the form of robust rate of movement.

The rate generates values according the following formula:

$$k(c_p, c_k, c_s) = \begin{cases} c_c & \text{when } c_p = c_k \text{ and } c_s \neq \text{" } - \text{"} \\ -c_c * 10 & \text{when } c_p = c_k \text{ and } c_s = \text{" } - \text{"} \\ -c_c & \text{when } c_p \neq c_k \text{ and } c_s \neq \text{" } - \text{"} \\ c_c * 10 & \text{when } c_p \neq c_k \text{ and } c_s = \text{" } - \text{"} \end{cases} \tag{8}$$

The combination of those functions leads to the general formula in the form as (9) and again, the highest score is selected, and that move is made.

C. The Hybrid Strategy

Both previous strategies have advantages and disadvantages of use. It can be noticed that the contrary of disadvantage in one strategy occurs as an advantage in the second strategy. The possible combination of both solutions gives a strategy with many advantages and few disadvantages. We call it the hybrid strategy.

This strategy like the previous strategies defines two functions: *position estimation function* and *movement estimation function*. Comparing the efficient-smart and hybrid strategies, there are two real differences: the definition of the latter function and the form of robust rate of movement. Now the measure of movement is given by the following formula:

$$f_d(C) = k(VP_p, c_p, c_k, c_s) * c_c \tag{9}$$

where

- $C = \{VP_p, c_p, c_k, c_s, c_c\}$ is a set attributes of card and player, where:
  - VP_p the number of victory points of player p,
  - c_p means colour of turtle belonging to the player,
  - c_k means colour of turtle belonging to the played card,
  - c_s means turtle movement type of the played card,
  - c_c means number of passed squares of the played card,
- $k(c_p, c_k, c_s)$ is the robust rate of movement.

In this case, the robust rate of movement produces values according (10) where VP_{max} means the number of victory points of leading player.

The combination of *position estimation function* and *movement estimation function* leads to the general formula in the form as (9) respectively, and again, the highest score is selected, and that move is made.

IV. Game Evaluation: Preliminary Results

In this section, some preliminary experimental results are given and a brief discussion is provided. One experiment consists in a match of 100 trial runs. The results allow to

$$k(VP_p, c_p, c_k, c_s) = \begin{cases} c_c * 10 & \text{when } c_p = c_k \text{ and } VP_{max} <= VP_p \text{ and } c_s \neq " - " \\ c_c & \text{when } c_p = c_k \text{ and } VP_{max} > VP_p \text{ and } c_s \neq " - " \\ -c_c * 10 & \text{when } c_p = c_k \text{ and } c_s = " - " \\ -c_c & \text{when } c_p \neq c_k \text{ and } c_s \neq " - " \\ c_c & \text{when } c_p \neq c_k \text{ and } VP_{max} <= VP_p \text{ and } c_s = " - " \\ c_c * 10 & \text{when } c_p \neq c_k \text{ and } VP_{max} > VP_p \text{ and } c_s = " - " \end{cases} \quad (10)$$

TABLE I: The percentage of wins in self-play performance by using MCTS for every combination of the strategies playing against each other.

	F	E	H	D
F	X	47	38	52
E	53	X	46	40
H	62	54	X	46
D	48	60	54	X

check, which strategy gives the best results for designated algorithms, and which gives the worst.

A. Monte Carlo Tree Search

MCTS is a recently proposed heuristic method that combines the precision of tree search with the generality of random sampling of the search space [14]. The most promising results so far have been obtained in the game of Go, in which it outperformed all classic techniques. Table I presents the results, when two opponents play against each other, both using MCTS for every combination of the strategies. Our results show that these values depend on the type of strategy. The percentage of wins is lower for fast (F) and efficient-smart (E) strategy, higher for dummy (D) and hybrid (H) strategy. Moreover, it should be noticed that using stronger strategy does not influence unambiguously on the winning performance.

B. Minimax Search

One way to pick the best move in zero-sum games is to search the game tree using the minimax algorithm [8]. Minimax is the method that minimizes the maximum possible loss. At each step of the game the assumption is made that player A is trying to maximize the chances of A's winning, while player B is trying to minimize the chances of A's winning. As shown in Table II, we can say that we observe a significant improvement for winning rates, even more than 30%. Both fast and hybrid strategies come above 80% of effectiveness over dummy strategy, what present their independence against randomness. The worst strategy is a dummy strategy. It gains below 25% odds of wining with others.

C. Alpha-beta Pruning

The most significant drawback of the minimax algorithm is that the whole game tree has to be searched. Since time requirements grow exponentially, for all practical purposes, the applied algorithm allows only a relatively small tree depth to be searched. A well-known approach to handle this problem is alpha-beta pruning [4]. It eliminates branches from being

TABLE II: The percentage of wins in self-play performance by using minimax search for every combination of the strategies playing against each other.

	F	E	H	D
F	X	53	46	81
E	47	X	43	78
H	54	57	X	80
D	19	22	20	X

TABLE III: The percentage of wins in self-play performance by using alpha-beta pruning for every combination of the strategies playing against each other.

	F	E	H	D
F	X	66	41	76
E	34	X	47	67
H	59	53	X	81
D	24	33	19	X

searched. The resulting move is always the same as in minimax search. So there is no loss of accuracy, but only improvements regarding the computational effort. In our case of multi-player sequential game of imperfect information, we must deal with probabilistic knowledge which is incomplete (risk assessment), and could be erroneous (possible deception). Lack of information can work in favour of the designated strategy but also against them. As summarized in Table III, again, the hybrid (H) and fast (F) strategies are over the others. For example, both mentioned strategies have about 75% of effectiveness over dummy strategy, where the maximum efficiency does not pass 35%.

D. Multiple Paired Algorithm and Strategy Comparison

Multiple paired comparison takes advantage of simple comparative judgements to prioritize a set of strategies, and is able to quantify the preferences of them. In the method of paired comparison, the basic measurement unit is the comparison of two methods. The results of paired comparisons can be summarized by a correlation matrix in our case, of the number of victories to the number of defeats for every combination of the strategies playing against each other. The comparison of MCTS versus alpha-beta pruning has been given in Table IV, MCTS versus minimax search in Table V, and alpha-beta pruning versus minimax search in Table VI respectively.

The analysis of experimental data for minimax search and alpha-beta pruning leads to observation of high congruity of received results for both algorithms by symmetrical assorted strategies. Thereupon the different operation algorithms do not

TABLE IV: The success rate by using MCTS vs. alpha-beta pruning for every combination of the strategies playing against each other.

		alpha-beta			
		F	E	H	D
MCTS	F	27:73	26:74	21:79	50:50
	E	29:71	27:73	18:82	57:43
	H	32:68	28:72	34:66	45:55
	D	20:80	27:73	24:76	53:47

TABLE V: The success rate by using MCTS vs. minimax search for every combination of the strategies playing against each other.

		minimax			
		F	E	H	D
MCTS	F	24:76	21:79	19:81	51:49
	E	27:73	28:72	37:63	51:49
	H	25:75	26:74	26:74	55:45
	D	24:76	23:77	32:68	52:48

TABLE VI: The success rate by using alpha-beta pruning vs. minimax search for every combination of the strategies playing against each other.

		minimax			
		F	E	H	D
alpha-beta	F	49:51	58:42	59:41	74:26
	E	42:58	58:42	36:64	74:26
	H	65:35	50:50	59:41	76:24
	D	20:80	21:79	26:74	42:58

provide a strong impact on game mechanics, which is mostly random. This means, that both algorithms for the same strategy get similar results. As a result of these observations, MCTS has the lowest effectiveness against the others, regardless of the strategy employed. The most likely reason for this discrepancy is the lowest computational complexity of the algorithm among these three tested. It should be emphasized that this approach often yields nonoptimal results that leaves it potentially most vulnerable to defeat.

V. Summary

Much of the effort put in this work has been aimed at developing *The Great Turtle Race* game program endowed with intelligent capabilities, based on a combination of three popular search techniques and four playing strategies. Based on the results of the experiments performed with our program, we may offer the following conclusions:

- In general, the simulation results are promising and suggest that further research needs to be performed. The best strategy proved to be the hybrid variant, the most difficult opponent to defeat in our game.
- By building on new deep learning techniques for board games of imperfect information, we can continue to make progress toward defeating a human expert player and the worst uncertainties by a computer program starting with a random initial configuration.
- To achieve the goal of satisfying strict team users, it is necessary to combine the best game strategies with collective intelligence concept. Adapting new algorithms to support mixed methods would improve the effectiveness of the opponent play, including anti-game strategy applied by more than one player, for example, involved in a ring.

Acknowledgment

This research received financial support from the statutory funds at the Wrocław University of Science and Technology, Poland.

References

[1] V. N. Marivate and T. Marwala, "Social learning methods in board game agents," in *2008 IEEE Symposium On Computational Intelligence and Games*, Dec 2008, pp. 323–328.

[2] M. Weber, V. Tirronen, and F. Neri, *Fitness Diversity Parallel Evolution Algorithms in the Turtle Race Game*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 303–312. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01129-0_34

[3] W. Jaśkowski, K. Krawiec, and B. Wieloch, "Evolving strategy for a probabilistic game of imperfect information using genetic programming," *Genetic Programming and Evolvable Machines*, vol. 9, no. 4, pp. 281–294, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10710-008-9062-1

[4] A. Saffidine, H. Finnsson, and M. Buro, "Alpha-beta pruning for games with simultaneous moves," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, ser. AAAI'12. AAAI Press, 2012, pp. 556–562. [Online]. Available: http://dl.acm.org/citation.cfm?id=2900728.2900808

[5] D. Strnad and N. Guid, "Parallel alpha-beta algorithm on the gpu," *Journal of Computing and Information Technology*, vol. 19, no. 4, pp. 269–274, 2011.

[6] C. Johnson, L. Barford, S. M. Dascalu, and F. C. Harris, *CUDA Implementation of Computer Go Game Tree Search*. Cham: Springer International Publishing, 2016, pp. 339–350. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-32467-8_31

[7] S. M. Lucas and G. Kendall, "Evolutionary computation and games," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 10–18, 2006.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[9] B. Arneson, R. B. Hayward, and P. Henderson, "Monte carlo tree search in hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[10] A. Papadopoulos, K. Toumpas, A. Chrysopoulos, and P. A. Mitkas, "Exploring optimization strategies in board game abalone for alpha-beta search," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 63–70.

[11] F. Maly, P. Kriz, and A. Mrazek, "An artificial player for a turn-based strategy game," in *Intelligent Information and Database Systems: 9th Asian Conference, ACIIDS 2017, Kanazawa, Japan, April 3-5, 2017, Proceedings, Part I*, N. T. Nguyen and et al., Eds. Springer International Publishing, 2017, pp. 455–465.

[12] D. Fudenberg and J. Tirole, *Game Theory*. Cambridge, MA: MIT Press, 1991.

[13] P. Hingston, M. Preuss, and D. Spierling, "Redtnet: A network model for strategy games," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–9.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, March 2012.