Journal of Cellular Automata, Vol. 0, pp. 1–21
Reprints available directly from the publisher
Photocopying permitted by license only

The Game of Go: A Cellular Automata Approach

JOSÉ MANUEL GÓMEZ SOTO* AND DIEGO DELGADO ÁVILA

Unidad Académica de Matemáticas, Universidad Autónoma de Zacatecas, Zacatecas, Zac. México
E-mail: diedelavi@gmail.com

Received: March 16, 2023. Accepted: March 20, 2023.

This paper shows a way to play the game of Go using cellular automata. To do this, we consider the best move depending on the player's style: offensive, defensive, or neutral. The best move is characterized by the importance of the next points: Chain growth, growth of degrees of freedom, reduction of opponent's degrees of freedom or capture of opponent's stones. We use cellular automata to compute these problems for each possible move and determine the next move. We also use cellular automata to evaluate the board at each moment and determine which player wins in the endgame.

Keywords: Cellular automata, game of Go

1 INTRODUCTION

Go (Chinese: Wei-ki, Korean: Baduk) is an ancient oriental game that originated in China over 4000 years ago. In 2017, the paper "Mastering the game of Go without human knowledge" [31] was published by David Silver, Julian Schrittwieser et al, describing a methodology for playing Go using Deep Neural Networks and Monte Carlo Tree Search (MCTS). Based on this methodology, a computer system called AlphaGo was developed that won the World championship against Lee Seedol in 2015.

Efforts to develop an algorithm to bear human players in the game of Go date back to 1960. In 1968, Albert Lindsey Zobrist [39] used pattern

* Corresponding author E-mail: jmgomez@uaz.edu.mx

recognition and hashing techniques to write the first Go program. Using IA techniques such as pattern recognition, human perceptions, and cognitive abilities, INTERIM later named NEMESIS was developed in 1979 by Walter Reitman and Bruce Wilcox [34]. Years later, NEMESIS was commercialized under the name Ego and would become the first program to compete in a tournament against human players [13]. The early development of Computer Go up to the late 1970s is described in more detail by Wilcox in [35].

In 1981, Jonathan K. Millen published an article in Byte Magazine about Wally [33], a program that plays Go, and has the ability to find the degrees of freedom of a chain and recognize some patterns. In the 80's of the last century, thanks to the tournaments that played Computational Go, the programs that play Go grew and Goliath was the program that dominated. Goliath was written by Mark Boon [2, 3] and also uses pattern recognition.

The first open code program GNU-Go, was written in 1989 and won several tournaments. The more popular version was "Game boy Advance", which was distributed by NINTENDO and sold 140,000 copies. In the decade of the 1990s, many competitions were held between computers and humans. Computer vs. human tournaments grew during this time and Go-Intellect [14] was one of the best computer programs. This is a very brief description about computer Go programs in those years, a more detailed description of the development of computer Go can be found in [4, 17, 24, 25].

Throughout this time, the most important methods have been rule-based systems, pattern recognition, knowledge base systems, specialized approaches to specific subproblems of Go, such as solvers for "life and death" [15, 36], capturing stone blocks [32], solving endgames [23], security of stones and territories [1, 21, 22, 26], and given solutions for seki [27]. But the method that had a great impact on the improvement of Go computer programs, was MonteCarlo Tree Search (MCTS) [5, 16]. At the beginning of the millennium, many computer Go programs integrated the MTCS and had many improvements, some of these programs were Fuego [10], Leela [18], Zen [38], MoGo [20], Crazy Stone [6], GNU-Go [11], the Many Faces of Go [9] and G++ [12].

Years later, another great method called Minorization-Maximization (MM) emerged to detect move patterns using Supervised Learning (SL) [7] and Deep Convolutional Neural Networks (DCNNs) [8, 19] that estimate how experienced human players respond with a prediction that exceeds the rate of previous methods. Alpha Go, the program that beat world champion Lee Seedol, combines MCTS with DCNN. The program uses MCTS in combination with three neural networks: a Supervised Learning network, a Reinforcement Learning network(RL), and a value network [29].

Recently, another more poweful system has emerged: AlphaZero Go, the successor of Alpha Go, is trained completely unsupervised and no domain

knowledge other than the game rules is implemented [30]. AlphaGo Zero defeated AlphaGo by 100 to 0 games.

This paper, explore a way to play Go using cellular automata. Cellular automata were developed by John von Neumann in the mid-1950s to show that machines can replicate themselves [37] and have many applications where changes in dynamical systems depend on local mappings [28].

Here, cellular automata are used to determine the best move on the Go board based on a local analysis that considers the growth of our chains, the degrees of freedom we can obtain, capturing opponent's stones or taking degrees of freedom from opponents. Cellular automata can also be used to evaluate the endgame and determine which player wins.

1.1 Cellular automata

A cellular automaton is a dynamical system in which time and space are discrete. The space consist of a set of cells, and each cell can have a state at any time in the evolution of the system. To obtain the state of a cell, a function is required that takes into account the states of the neighboring cells at the previous time step. Formally is defined as:

Definition 1.1. *Let be a group G and a set A. A cellular automaton over the group G and the alphabet A is a mapping $\Phi : A^G \to A^G$, satisfying the following property: $\exists\, N = \{x_0, x_1, \ldots, x_{n-1}\} \subset G$ and a mapping $\phi : A^N \to A$, denotated by $\phi(h) = \phi(y_0, y_1, \ldots, y_{n-1})$ with $y_i = h(x_i)$, such that $\forall\, s \in A^G, \forall\, x \in G$:*

$$\Phi(s)(x) = \phi(s(x + x_0), s(x + x_1), \ldots, s(x + x_{n-1})).$$

Note that Φ is determined by the local rule ϕ. When we talk about cellular automata in this paper, we will use the local rule as follows:

$$s_{t+1}(x) = \phi(s_t(x + x_0), s_t(x + x_1), \ldots s_t(x + x_{n-1})).$$

To consider a cellular automaton, we contemplate a square grid where each cell represents the crossing of lines on the board. In this way, we consider the group $G_n = Z_{n+1}^2$ with board size $n \times n$, where in this group the spaces in $\{0\} \times Z_{n+1}$ y $Z_{n+1} \times \{0\}$ represent the edges of the board (see the Figure 1).

2 EVALUATION OF GO BY CELLULAR AUTOMATA

The evaluation of the board determines the territory reached by each player and his corresponding stones. We use the Chinese evaluation method, which

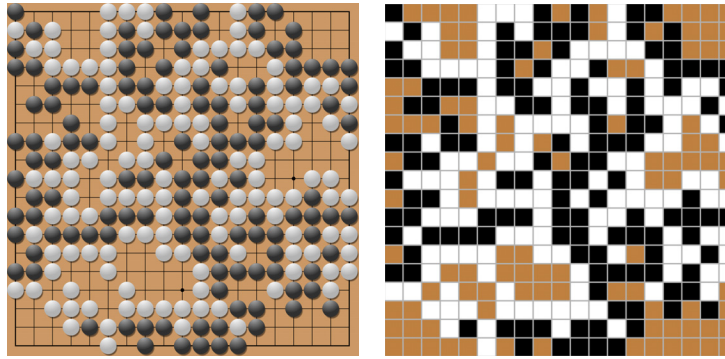


FIGURE 1
The crossing of lines on the Go board is represented by a cell in the cellular automaton. Thus, the intersection of the lines occupied by a black stone is a black cell, the intersection of the lines occupied by a white stone is a white cell, and when the intersection of the lines is empty, it is represented by a brown cell.

consists in counting the stones of a color and the spaces surrounding the stones of that color (territory). This was done for each color and the color that has more territories and stones wins. Figure 2 illustrates this type of evaluation. In the case where both colors have the same number, the white stone wins because the black stone opens the game.

2.1 Cellular automata of evaluation

As mentioned earlier, we consider a cellular automaton as a square grid where each cell represents the intersection of lines on the board. In this way, consider the group $G_n = Z_{n+1}^2$ with board size $n \times n$, in this group the spaces in $\{0\} \times Z_{n+1}$ y $Z_{n+1} \times \{0\}$ represent the edges of the board. And since the intersecting lines are orthogonally connected, the Von Neumann neighborhood is considered to represent this connection (see Figure 3).

Definition 2.1 (Von Neumann's neighborhood). *The von Neumann's neighborhood at a position $x = \{i, j\} \in G_n$ is defined by:*

$$N_x = \{(k, l) \in G_n : |k - i| + |l - j| \leq 1\}.$$

We used cellular automata to evaluate a Go board. By conveniently defining the function ϕ, we can show that the evolution of cellular automata converges to a fixed point, the configuration that determines the spaces and stones belonging to each player and, moreover, can determine who wins. In this way, the cellular automaton for evaluating the board is defined as follows:

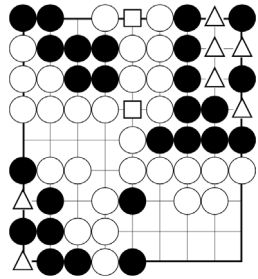


FIGURE 2
In this example, we mark the territory belonging to the black stones with a triangle, and the territory belonging to the white stones with a square. So, if we add the black stones plus the triangle, we have 33, and if we add the white stones plus the square, we have 30. The black stones win.

Definition 2.2 (Cellular Automata of Evaluation). *The cellular automata over the group* $G_n = \mathbb{Z}_{n+1}^2$ *and the alphabet* $A = \{-1, 1, 2, 3, 6, 10, 15\} \subset \mathbb{Z}$ *and the Von Neumann neighborhood* N_x*. These cellular automata are called Cellular Automata of Evaluation if they have the next local rule:*

$$s_{t+1}(x) = \begin{cases} -1 \text{ if } s_t(x) = -1 \\ 15 \text{ if } s_t(x) = 15 \\ 10 \text{ if } s_t(x) = 10 \\ 6 \text{ if } (\prod_{y \in N_x} s_t(y)) \mid 6 \\ 3 \text{ if } (\prod_{y \in N_x} s_t(y)) \mid 3 \\ 2 \text{ if } (\prod_{y \in N_x} s_t(y)) \mid 2 \\ 1 \text{ other case.} \end{cases}$$

Here we denote -1 for the edge board, 1 for free, 10 for the white stone, 2 for the cells belonging to the white stones, 15 for the black stone, 3 for the cells belonging to the black stone, and 6 for the cells not belonging to the black or white stones.

The logic behind the local rule is, if we have a black stone, it remains as a black stone, the same is true for the white stone. We consider a place indeterminated, if the cells of the neighborhood do not belong to a black stone or a black territory and do not belong to a white stone or a white territory. The cell is a black territory if it is surrounded by a black stone or a black territory. Similarly, the cell is a white territory if it is surrounded by a white stone or a white territory. In other cases, the cell is considered free.

Cellular automata for board evaluation characterize the territory of black and white stones. Depending on the local neighborhood, at each step of the

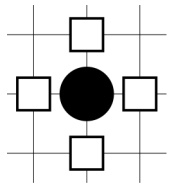


FIGURE 3
Von Neumann's neighborhood

cellular automata evolution each cell is determined to be free, black stone, white stone, black territory, or white territory, until no more changes are possibles in the cellular automata's evolution, i.e., until the cellular automata reaches its fixed point. Once the fixed point is reached, the black stones and and their territory are counted, and the same is done for the white stones. In this way, the board is evaluated.

The evolution of the cellular automaton can be considered as an orbit $x, \Phi(x), \Phi^2(x), \Phi^3(x),\ldots, \Phi^3(x)$ where $\Phi^2(x) = \Phi(\Phi(x))$, $\Phi^3(x) = \Phi(\Phi(\Phi(x)))$,..., $\Phi^n(x) = \Phi(\Phi(\Phi(...)))$ n times. In this context, we define the fixed point as follows:

Definition 2.3 (Fixed point). *Given a cellular automaton Φ over the group G and the alphabet A, a fixed point x of Φ is a mapping $x \in A^G$ such that:* $\Phi(x) = x$.

Then we can show from an initial configuration that the evolution converges to its fixed point, and this configuration will show us which territory belong to black stones and white stones (See Figure 4). The proof of the next proposition shows that the cellular automata of evaluation converge to their fixed point.

Proposition 2.4. *Given a cellular automata of evaluation, $\forall\, s_0 \in A^G$, $\exists\, t \in \mathbb{N}$. Then exist a fixed point, it means that:*

$$\Phi(s_t) = s_{t+1} = s_t.$$

Proof. Given $s_0 \in A^{G_n}$, $B_i = \{x \in G_n : s_i(x) \mid 3\}$ and $W_i = \{x \in G_n : s_i(x) \mid 2\}$. We will show that if $x \in B_i$ then $x \in B_{i+1}$.

Case 1. If $s_i(x) = 15$ then $s_{i+1}(x) = 15$. In this way $x \in B_{i+1}$.

Case 2. If $s_i(x) = 6$, then $\prod_{y\in N} s_i(x+y) = 6\left(\prod_{y\in N\setminus 0} s_i(x+y)\right)$, imply $\left(\prod_{y\in N} s_i(x+y)\right) \mid 6$, in consequence $s_{i+1}(x) = 6$, therefore $x \in B_{i+1}$.

Case 3. If $s_i(x) = 3$, then $\prod_{y\in N} s_i(x+y) = 3\left(\prod_{y\in N\setminus 0} s_i(x+y)\right)$, imply $\left(\prod_{y\in N} s_i(x+y)\right) \mid 3$, in consequence $s_{i+1}(x)|3$, therefore $x \in B_{i+1}$.

In this way $B_i \subset B_{i+1}$.

Then because $B_i \subset G_n$, $\forall\, n \in \mathbb{N}$ and G_n is a finite group, then $\exists\, t' \in \mathbb{N}$, $B_{t'} = B_{t'+m}$, $\forall m \in \mathbb{N}$. And the analogous apply to W_i. Let $t' \in \mathbb{N}$, this satisfy both, B_i and W_i.

Let $x \in G_n$, if $s_0(x) = -1$ then $\forall\, i \in \mathbb{N}$, $s_i(x) = -1$. If $s_0(x) = 1$ then $\forall\, i \in \mathbb{N}$, $s_i(x) = 1$ o $\exists\, i \in \mathbb{N}$ such that $s_i(x) > 1$, in this case $x \in B_i \cup W_i$, imply $s_{t'}(x) = s_{t'+m}(x)$, $\forall\, m \in \mathbb{N}$.

Furthemore $\forall\, x \in G_n, \exists\, t_1 \in \mathbb{N}$ such that $s_{t_1}(x) = s_{t_1+m}(x)$, $\forall\, m \in \mathbb{N}$. Then because G_n is finite, $\exists\, t \in \mathbb{N}$ such that $s_t = s_{t+1}$. Therefore the Cellular Automata of Evaluation has a fixed point. □

3 GAME'S VECTOR

The style of the player is characterized by a game-vector and a move-vector. The first one consists of elements that determine whether a player acts offensively, defensively or otherwise. The move-vector determines the value of each cell of the game board, depending on whether the player is offensive or defensive or otherwise. First we define the game-vector:

3.1 Game-vector

Let us define the game-vector:

Definition 3.1 (game-vector). *We define the game-vector as:*

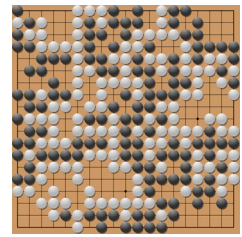
$$v = [d_1, d_2, o_1, o_2]$$

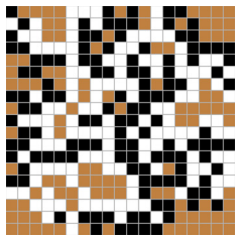
where:

- d_1*: Grow chain.*
- d_2*: Growth of degrees of freedom.*
- o_1*: Decrease the opponent's degrees of freedom.*
- o_2*: Capture of opponent's stones.*

y $d_1, d_2, o_1, o_2 \in \mathbb{Z}$.

High values of d_1, d_2 with respect to o_1, o_2 are considered an aggressive style, and high values of o_1, o_2 with respect to d_1, d_2 are considered a defensive style. A game-vector where d_1, d_2 has negative values is considered

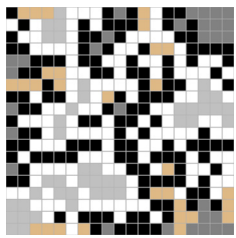




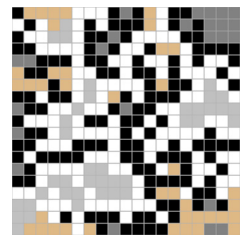
x

$\Phi^2(x)$

$\Phi^2(x)$

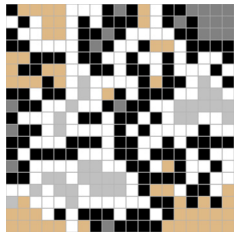


$\Phi^3(x)$

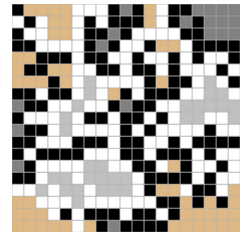


$\Phi^4(x)$

$\Phi^5(x)$



$\Phi^6(x)$



$\Phi^7(x)$

$\Phi^8(x)$

FIGURE 4
This is an example of how the cellular automata of evaluation work. From the initial configuration and its respective CA configuration, it is possible to see the evolution or orbit $x, \Phi(x)$, $\Phi^2(x)$, $\Phi^3(x)$, $\Phi^4(x)$, $\Phi^5(x)$, $\Phi^6(x)$, $\Phi^7(x)$, $\Phi^8(x)$ and how each cell is determined until it reaches the fixed point i.e. $\Phi^7(x) = \Phi^8(x)$. In this example we have represented the states by colors, 1: brown (free), 2: Ligth gray (white territory), 3: gray (black territory), 6: Light brown (indeterminate), 10: white and 15: black. In the fixed point configuration, counting the white territory (white and light gray cells) is $132 + 31 = 163$ and counting black territory (black and gray cells) is $127 + 19 = 146$, therefore white win.

"other",i.e. it has no interest in expanding its chain or increasing its degrees of freedom and is either not interested in decreased degrees of freedom or capturing stones from its opponent.

Now, to know which is the best move, we need to know what the move means in the context of *chain growth*, *increased degrees of freedom*, *reduction of opponent's degrees of freedom*, and *capturing opponent's stones*. Furthermore, for each board position we define a vector called *move-vector* defined in the next section.

4 MOVE'S VECTOR

Each position of the board is characterized by a move-vector. This vector has the same parameters as the game-vector, but its value is assigned according to the situation of the free position on the board. In this way, each free position on the board has a move-vector.

This vector is defined as:

4.1 Move-vector

Definition 4.1 (move-vector). *We define the move-vector as:*

$$m = [m_1, m_2, m_3, m_4]$$

where:

- m_1*: grow chain.*
  If the move makes that chain grow then it has the value 1, if the move join n chains then it get the value n, and if the move create a new chain, will get the value 0.
- m_2*: Grow degrees of freedom.*
  If the move creates a new chain it gets the value 1. In all other cases the score is the difference between the degrees of freedom after and before the move.
- m_3*: Decrease of the opponent's degrees of freedom.*
  The difference between the degrees of freedom of the opponent's stones after the move and before the move.
- m_4*: Capture of opponent's stones.*
  The length of the captured chains.

y $m_1, m_2, m_3, m_4 \in \mathbb{Z}$*.*

The game-vector and the move-vector have the same parameters, but the difference is that the game-vector characterizes the player and the move-vector characterizes any free position on the board. Both vectors help

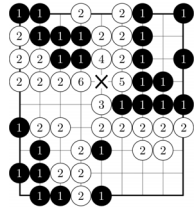


FIGURE 5
Convenient initial conditions from which the cellular automata of the chain characterizes the chains when the white stone moves in the × position.

make a decision about the next move. The parameters of the game-vector are randomly assigned and the parameters of the move-vector are computed by the evolution of cellular automaton. In the following sections is shown how the individual parameters of the move-vector are calculated.

4.1.1 Parameter m_1

In the move-vector, the parameter m_1 is about chain growth. If the move allows the growth of chain, it has the value 1 and if the move connects n chains, it has the value n.

To compute this, we use a cellular automaton called *Cellular Automata of Chain*, defined as follows:

Definition 4.2 (Cellular Automata of Chain). *The cellular automaton over the group $G_n = \mathbb{Z}_{n+1}^2$ and the alphabet $A = \{-1, 1, 2, 3, 4, 5, 6\} \in \mathbb{Z}$ and the Von Neumann neighborhood N_x. These cellular automata are called Cellular Automata of Chain if they have the next local rule:*

$$s_{t+1}(x) = \begin{cases} -1 \text{ if } s_t(x) = -1 \\ max_{y \in N_x}(s_t(y)) \text{ if } s_t(x) \neq 1 \\ 1 \text{ other case.} \end{cases}$$

To use this cellular automaton, we define the initial configuration in a particular way. The allied stones are assigned the value 2 and all others are assigned the value 1*, except for the allied stones surrounding the position where the next move is to occur i.e., their neighbor with the value 2 will have values greater than 1 and 2. To illustrate this, consider the example of Figure 5, where we give the allied stones surrounding the move the values 3, 4, 5, or 6† . Note that the free position also has state 2, but for clarity we mark this with a cross.

* Opponent stones are not considered because we want to study allied stones; in this way, opponent stones are considered as free spaces with value 1.

† We mention only 4 values because this is the maximum number of neighbors in a von Neumann neighborhood without a central cell.

Since the local rule has the obligation to consider the maximum value of its neighbor, the chains connected to the position assume the state of the neighbor that connects them to the chain. All chains have different states when the cellular automaton arrives at its fixed point (see Figure 6). If two or more neighbors are connected to the same chain, it is characterized by the maximum value.

The proof of the next proposition shows that the evolution of the cellular automata of chain always converges to a fixed point, starting from any initial condition.

Proposition 4.3. *Given the cellular automata of chain,* $\forall\, s_0 \in A^G, \exists\, t \in \mathbb{N}$. *Then there exists a fixed point, that is:*

$$\Phi(s_t) = s_{t+1} = s_t.$$

Proof. Give $\Phi : A^G \to A^G$ cellular automata of chains over G_n with alphabet A, let $s_0 \in A^G$ and $x \in G_n$ then, for some $t \in \mathbb{N}$, if $s_t(x) = 1$ then $s_{t+1}(x) = 1$ and if $s_t(x) \neq 1$ then $s_{t+1}(x) \geq s_t(x)$ additionaly that $s_t(x) \in A$, $s_{t+1}(x) \in A$; then, because A is finite, the sequence $\{s_i(x)\}$ arrive to a fixed point $\forall x \in G_n$, and then, because G_n is finite, the cellular automata of the chain have a fixed point. □

4.1.2 Parameter m_2

The goal of the parameter m_2 is to know how many degrees of freedom are obtained after a move. In this way, the parameter m_2 is the difference between the number of degrees of freedom after the move at position $\times$ and the number of degrees of freedom before the move at position $\times$ (see the figure 7).

To do this, in each step of the cellular automata evolution we identify the cells that have degrees of freedom because they surround the allied stones (state 4 -green-), and at the same time we identify the cells that are not yet defined as degrees of freedom (state 8 -blue-) and the cells that cannot be free in any way (state 6 -red-), because they belong to the black or white territory. During the evolution of cellular Automaton, there is a struggle between state 6 (red) and state 4 (green), which try to get the cells not yet defined (state 8 - blue), which state will win depends on whether the chain of cells not yet defined (state 8 -blue-) has a free place at the edge or not. In the end, the sum of the degrees of freedom will be the cells with state 4.

Formally, this local rule is defined as follows:

Definition 4.4 (Cellular Automata of degrees of freedom). *The cellular automata over the group* $G_n = \mathbb{Z}^2_{n+1}$, *and the alphabet* $A = \{-1, 1, 2, 3, 4, 6, 8\} \in \mathbb{Z}$ *and the Von Neumann neighborhood* N_x. *These*

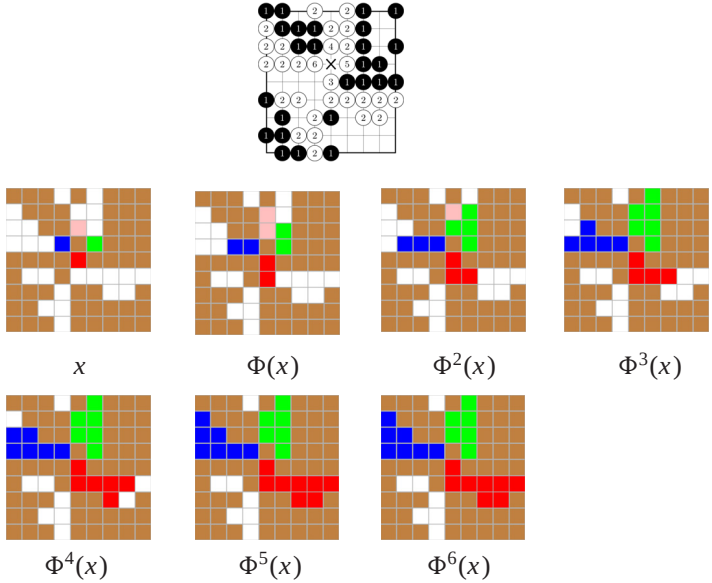


FIGURE 6
In this figure we can see how the cellular automata of chain distinguish the chains connecting the $\times$ position, when the white stone plays in that position. The chains are fully characterized when the cellular automata of chain reach their fixed point, i.e., when $\Phi^5(x) = \Phi^6(x)$. In this example, we identify state 1 with the color brown, state 2 with white, state 3 with red, state 4 with pink, state 5 with green and state 6 with blue. At the end we can distinguish three chains with blue, red and green color.

cellular automata are be called Cellular Automata of degrees of freedom if it has the next local rule:

$$s_{t+1}(x) = \begin{cases} -1 \text{ if } s_t(x) = -1 \\ 2 \text{ if } s_t(x) = 2 \\ 6 \text{ if } s_t(x) \in \{3, 8, 6\}, s_t(y) \in \{1, 6, 4\} \text{ to some } y \in N_x \\ 8 \text{ if } s_t(x) \in \{3, 8\}, s_t(y) = 2 \text{ to some } y \in N_x \\ 3 \text{ if } s_t(x) = 3 \\ 4 \text{ if } s_t(x) \in \{1, 4\}, s_t(y) = 2 \text{ to some } y \in N_x \\ 1 \text{ other case.} \end{cases}$$

The proof of the next proposition shows that the evolution of the cellular automata of degree of freedom always converges to a fixed point, starting from any initial condition.

Proposition 4.5. *Given a cellular automaton of degree of freedom* $\forall s_0 \in A^G, \exists t \in \mathbb{N}$. *Then let there exist a fixed point, it means that:*

$$\Phi(s_t) = s_{t+1} = s_t.$$

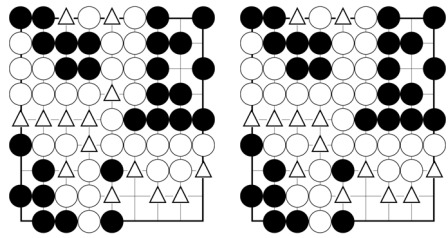


FIGURE 7
Is an example where $m_2 = -1$, given the difference between the number of degrees of freedom after (14 degrees of freedom) and before the move at position $\times$ (13 degrees of freedom), $m_2 = 13 - 14 = -1$ The positions that belong to degrees of freedom are marked as triangles.

Proof. Let s_0 be such an a initial configuration that $s_0(w) \in \{1, 2, 3\}$, $\forall w \in G$.
Let $S_a^t = \{x \in G : s_t(x) = a\} \subset G$.

Case 1. Let $x \in S_6^t$, $s_t(x) = 6$ and $\exists\, y = x \in N_x$ such that $s_t(y) \in \{1, 6\}$, in this way $s_{t+1}(x) = 6$ it follow that $x \in S_6^{t+1}$, then $S_6^t \subset S_6^{t+1}$, and because G is finite, the sequence $\left\{S_6^i\right\}$ has a fixed point, and this is founded at time $k \in \mathbb{N}$.

Case 2. Let $x \in S_8^k$, $s_k(x) = 8$ and because $s_0(w) \neq 8$, $\forall w \in G$ then $\exists\, y \in N_x$ such that $S_k(y) = 2$, therefore $s_{k+1}(x) = 8$, because if $s_{k+1}(x) = 6$ it is a contradiction about that k is a fixed point of the sequence $\left\{S_6^i\right\}$. In this way, and analogously to case 1, the sequence $\left\{S_8^i\right\}$ has a fixed point.

Case 3. If $x \in S_3^{t+1}$ then $x \in S_3^t$, so $S_3^{t+1} \subset S_3^t$ and since they are finite and discrete sets, the sequence $\left\{S_3^i\right\}$ has a fixed point.

Case 4. Let $x \in S_4^t$, because $s_0(x) \neq 4$, $\forall w \in G$, then $\exists\, y \in N_x$ such that $s_t(y) = 2$, in this way $s_{t+1}(x) = 4$, furthermore $S_4^t \subset S_4^{t+1} \subset G_n$ y $\left\{S_4^1\right\}$ has a fixed point.

Case 5. Let $x \in S_2^t$, then $s_0(x) = 2$, because taking the definition, to $t > 0$, $s_t(x) = 2$ if and only if $s_{t-1}(x) = 2$, furthemore $\left\{S_2^i\right\}$ is a constant, it follow that it has a fixed point.

Case 6. In analogous way as the early case the sequence $\left\{S_{-1}^i\right\}$ has a fixed point.

Case 7. As $S_1^t = G_n - \bigcup_{a \in A \setminus \{1\}} S_a^t$, the sequence $\left\{S_1^i\right\}$ has a fixed point.

Furthemore, the cellular automata of degrees freedom has a fixed point. □

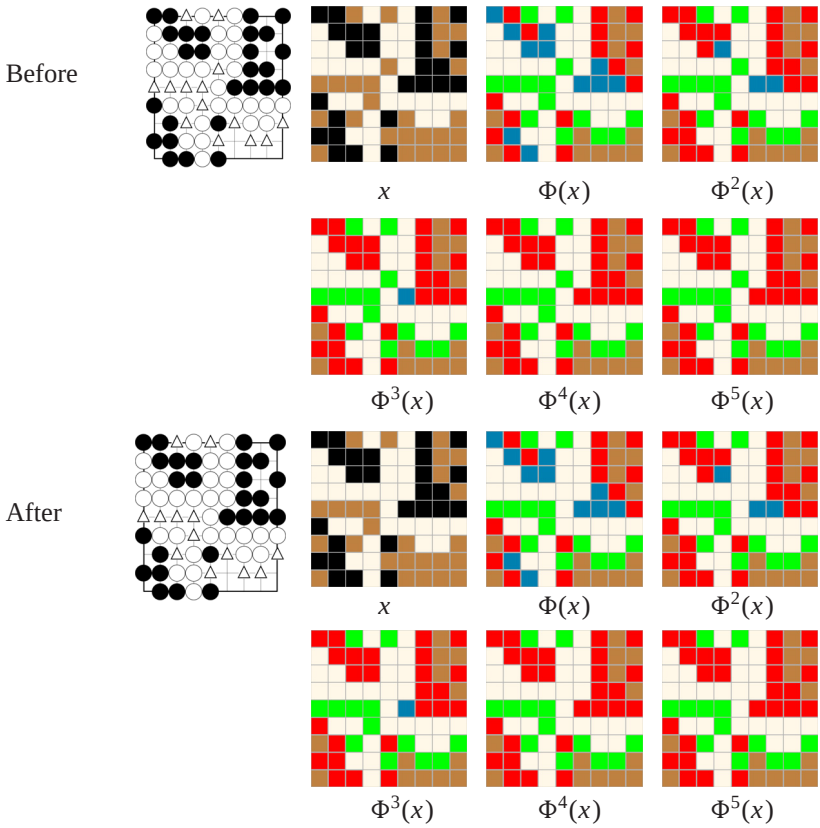


FIGURE 8
In this figure we show the example of Figure 7 in terms of the evolution of cellular automata. The degrees of freedom at the fixed point $\Phi^4(x) = \Phi^5(x)$ before the move vs the degrees of freedom at the fixed point $\Phi^4(x) = \Phi^5(x)$ after the move. Here we assign brown color to state 1 (free position), white color to state 2 (white stone), black color to state 3 (black stone), green color to state 4 (degree of freedom), red color to state 6 that belonging to the black or white territory (cells that cannot be degrees of freedom), blue color to state 8 (cell not yet determined). Note that in the first line we show the evolution from the initial condition without a move, we called *Before*, from its fixed point we can count 14 positions as degrees of freedom, and in the second line *After* the evolution from an initial configuration in which the move is performed. In the fixed point of the second line we can count 13 degrees of freedom. If we then set off the degrees of freedom after evolution with the degrees of freedom before evolution, we get $m_2 = 13 - 14 = -1$

To ilustrate how the Cellular Automata of degrees freedom work, see the Figure 8.

4.1.3 Parameter m_3

The goal of the parameter m_3 is to know how many degrees of freedom are taken from the opponent after the move of the allied stones. In this way, the

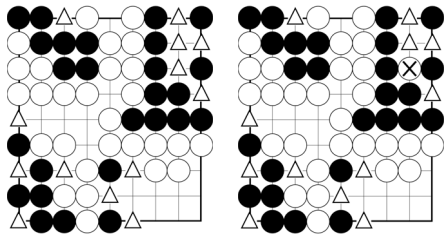


FIGURE 9
This is an example where $m_3 = -1$ because the difference between the number of degrees of freedom of the opponent after (12 degrees of freedom) and before moving to position $\times$ (13 degrees of freedom) $m_3 = 12 - 13 = -1$. The positions of the degrees of freedom are indicated by a triangle.

parameter m_3 is the difference between the number of opponent degrees of freedom after the move in position $\times$ and the number of opponent degrees of freedom before the move of the allied stones in position $\times$ (See the Figure 9).

The parameter m_3 is computed by the same cellular automaton as the parameter m_2, i.e., by the cellular automaton of degrees of freedom, but with a small difference: the opposing stones are now allies. Therefore, this cellular automaton has a fixed point, as shown in proposition 4.5.

Figure 10 shows how the cellular automata of degrees freedom work to compute m_3.

4.1.4 Parameter m_4

In the move-vector, the parameter m_4 is the number of stones captured from the opponent. Captured stones are groups or single stones that lose degrees of freedom due to the move of the opponent's stones. The Figure 11 shows an example of the stones that are considered captured.

We will use a cellular automaton to compute how to identify the captured stones. The idea behind the local rule is the following: If the opponent's stone is surrounded by free space, it cannot be captured. If this opponent does not have a free space as a neighbor, it is captured. In this way, at the end of the evolution, we count the captured stones from the opponent. Moreover, the border is kept as a border and the allied stones remain as allied stones. The cellular automata that captured stones is defined as *Cellular Automata of Capture*:

Definition 4.6 (Cellular Automata of Capture). *The Cellular Automata over the group $G_n = \mathbb{Z}^2_{n+1}$, and the alphabet $A = \{-1, 1, 2, 3, 4\} \in \mathbb{Z}$ and the Von Neumann's neighborhood N_x. This cellular automata will be called*

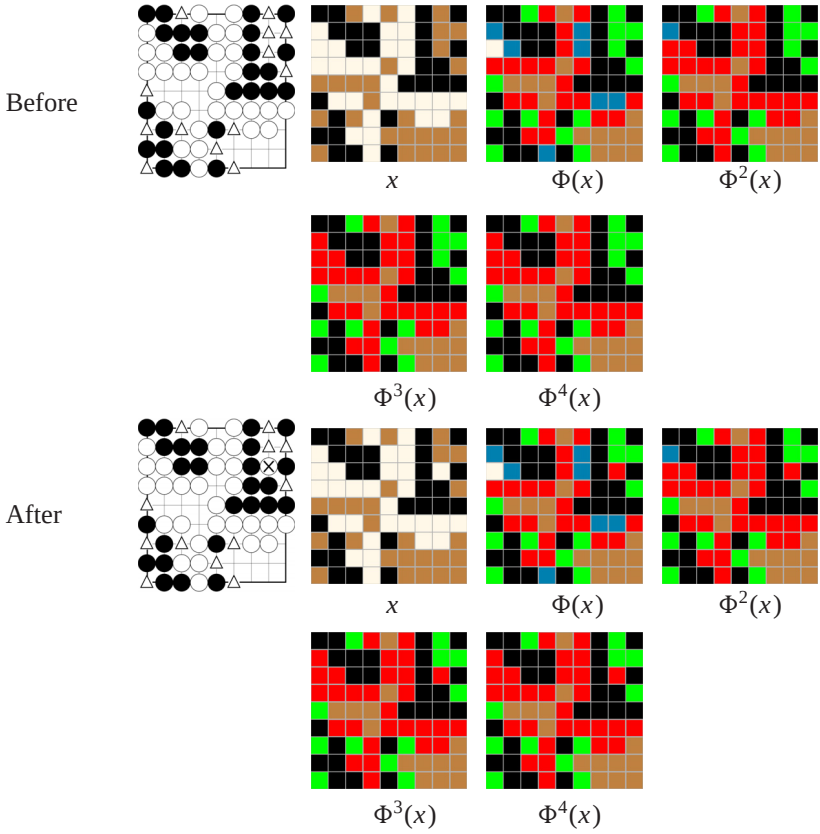


FIGURE 10
In this figure, we show the example of Figure 9 in terms of the evolution of cellular automata. The opponent's degrees of freedom at the fixed point $\Phi^3(x) = \Phi^4(x)$ before the move vs the opponent's degree of freedom at the fixed point $\Phi^3(x) = \Phi^4(x)$ after the move. Here we assign a brown color to state 1 (free position), a white color to state 3 (white stone), a black color to state 2 (black stone), a green color to state 4 (degrees of freedom), a red color to state 6 that belonging to the black or white territory (cells that cannot be degrees of freedom), a blue color to state 8 (cell not yet determined). Note that in the first line we show the evolution from the initial condition without a move, which we called it *Before* from its fixed point we count 12 positions as opponent degrees of freedom and in the second line *After* the evolution from an initial configuration where the move is performed. In the fixed point of the second line we can count 13 opponent degrees of freedom. If then we count the opponent's degrees of freedom after the evolution againts opponent's degrees of freedom before evolution, we get $m_3 = 12 - 13 = -1$.

Cellular Automata of Capture if has the next local rule:

$$s_{t+1}(x) = \begin{cases} -1 \; if \; s_t(x) = -1 \\ 2 \; if \; s_t(x) = 2 \\ max_{y \in N_x}(s_t(y)). \end{cases}$$

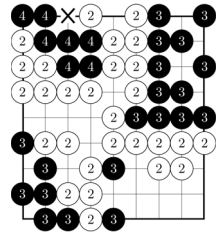


FIGURE 11
The white stones have the state 2 and the black stones have the state 3, and the captured black stones are with the state 4 after the move of the white stones in the position $\times$.

Where state -1 represents the edge of the board, 1 the free position, state 2 the white stones, 3 the black stones, and state 4 the captured stones.

The proof of the next proposition shows that the evolution of cellular automata of captures always converges to a fixed point, from any initial condition.

Proposition 4.7. *Given a Cellular Automata of Capture,* $\forall\, s_0 \in A^G, \exists\, t \in \mathbb{N}$. *Then exist a fixed point, it means that:*

$$\Phi(s_t) = s_{t+1} = s_t.$$

Proof. Let $x \in G_n$ and $t \in \mathbb{N}$, and by virtue of the definition of cellular automata of capture, $s_t(x) \leq s_{t+1}(x) \in A$, and since A is finite the sequence $\{s_i(x)\}$ has a fixed point. Since G_n is finite, the cellular automaton of capture also has a fixed point. □

Figure 12 illustrates how the cellular automata of capture works.

Up this point we have defined game-vector and move-vector. Now we will use both of them to characterize each free position of the board in order to choose the best move.

5 STRATEGY TO PLAY

For the selection of the best move, we chose a method that requires only the knowledge of the move that several game-vectors would make. This strategy consists of randomly selecting a set of game vectors, calculating which move each of them would make, and making the move that turns out to be most among the results.

The algorithm is as follows:

1. A random sample of game-vectors is generated.
2. With each vector of the sample, the evaluation of the moves is made to build the set of moves from which the move is selected. We create this set in this way:
   In order to know which is the best move, we define a function that evaluates the free positions and build a set of the best moves, from which we obtain the position to be played. For this purpose, we will assign a value to each position using the game-vector and the move-vector. The free position with the largest value will be considered the best play option. If two or more moves are equally good, they will build a set of the best moves from which we will choose the best option. Next we will explain in detail how to determine the best move.

   Definition 5.1 (Set of best moves). *Let be a strategic vector v, a board $T_n = G_n \setminus ((\mathbb{Z}_{n+1} \times \{0\}) \cup (\{0\} \times \mathbb{Z}_{n+1}))$, the set of white stones W and the set of black stones B, let $F = T_n \setminus (W \cup B)$.*
   The set of best moves is defined as:

$$M_v(F) = \{p \in F : \varphi_v(q) \leq \varphi_v(p), \forall q \in F\}$$

   Where $\varphi_v(p) = v_p \cdot v$ y v_p is the move-vector of p.

   Considering the set of best moves in this way, the best move given a move-vector v, is obtained as:

   (a) Let $p \in M_v(F)$.
   (b) If $\varphi_v(p) < 0$ we pass to the next move.
   (c) If p means -suicide-, go back step 1 with $F := F \setminus \{p\}$.
   (d) If p means -ko-, go back step 1 with $F := F \setminus \{p\}$.
   (e) Other case play p.
3. The best move will the most frequently best ranked.

6 CONCLUSION

This paper shows a way to play the game of Go using Cellular Automata. To do this, we consider the best move according to the player's style: offensive, defensive, or neutral, and characterize the free positions on the board considering the grown chains, the grown degrees of freedom, the reduction of the opponent's degrees of freedom, and the captures of the opponent's stones. We use the strategy to chosen the best move as the most frequently best ranked; however, other strategies can be used, such as genetic algorithms, statisti-

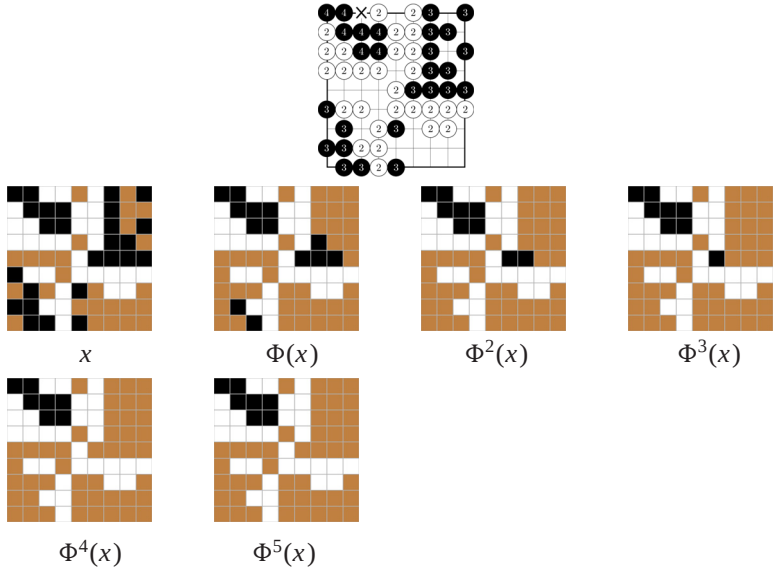


FIGURE 12
In this figure, we can see how cellular automata of capture caracterize the captured stones when a white stone moves to $\times$ position. In the initial condition, the allied stones have the white color, the opponent stones have the black color, and the free spaces have a brown color. In the first generation, the black stone is considered a candidate for capture. If the black stones has a free degree as a neighbor (brown color), it cannot be captured and is considered a free space. The stone to be captured (black color) that does not have a free cell as a neighbor (brown color) is considered to be captured and remains in this state. At the end, when cellular automata of capture reaches its fixed point $\Phi^4(x) = \Phi^5(x)$, the captured stones are marked with black color and are ready to be counted.

cal methods, etc., all based on cellular automata that characterize the free positions on the board. We implemented our strategy in the Julia programming language and play games on the Online Go Server (https://www.online-go.com) where our program was rated as a 22 kyu player.

REFERENCES

[1] D. Benson. "Life in the game of Go," Information Sciences, 10:17–29, 1976. Reprinted in Computer Games, Levy, D.N.L. (Editor), Vol. II, pp. 203–213, Springer Verlag, New York 1988.

[2] Mark Boon. (1989). "A pattern matcher for Goliath," Computer go, 13:12–23.

[3] Mark Boon. (1990). "A pattern matcher for Goliath," Computer Go, 13, 12–23.

[4] B. Bouzy and T. Cazenave. (2001). "Computer Go: An AI-oriented survey," Artificial Intelligence, 132(1):39–103.

[5] R. Coulom. (2006). “Efficient selectivity and backup operators in Monte-Carlo tree search,” in H. Jaap van den Herik, P. Ciancarini, and H. H. L. M. (Jeroen) Donkers (editors), Computers and Games, Berlin, Heidelberg, Springer, 72–83.

[6] Rémi Coulom, “Monte-CarloTreeSearchinCrazyStone,” Proc.GameProg. Workshop, Tokyo, Japan (2007).

[7] R. Coulom. (2007). “Computing Elo ratings of move patterns in the game of Go,” in Proceeding of Computer Games Workshop 2007 (CGW 2007), Amsterdam, The Netherlands, June 15–17, 113–124.

[8] C. Clark and A. Storkey. “Teaching deep convolutional neural networks to play Go,” December (2014), [Online] Available: http://arxiv.org/abs/1412.3409.

[9] David Fotland. “Knowledge Representation in The Many Faces of Go,” February (1993). https://www.smart-games.com/knowpap.txt., date retrieved: March 3, 2023.

[10] Fuego. http://fuego.sourceforge.net, date retrieved: Apr 13, (2021).

[11] GNU Go. https://www.gnu.org/software/gnugo/, October 2019. Consulted in April 3, (2021).

[12] History of Go-playing Programs, 2018. https://www.britgo.org/computergo/history, date retrieved: April 13, (2021).

[13] History of Go-playing Programs — British Go Association. https://www.britgo.org/computergo/history, Date retrieved: May 09, (2021).

[14] Ken Chen y Zhixing Chen. (1999). “Static analysis of life and death in the game of Go,” Information Sciences, 121:113–134.

[15] A. Kishimoto and M. Müller. (2005). “Search versus knowledge for solving life and death problems in Go,” Twentieth National Conference on Artificial Intelligence (AAAI-05), 1374–1379.

[16] L. Kocsis and C. Szepesvári. (2006). “Bandit based Monte-Carlo planning,” Proceeding of 17th European Conference on Machine Learning (ECML 2006), Berlin, Germany, September 18–22, 282–293.

[17] C. -S. Lee et al. (2016). “Human vs. Computer Go: Review and Prospect [Discussion Forum],” IEEE Computational Intelligence Magazine, 11(3):67–72, doi: 10.1109/MCI.2016.2572559.

[18] Leela. https://www.sjeng.org/leela.html, date retrieved: April 13, (2021).

[19] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver. “Move evaluation in Go using deep convolutional neural networks,” Dec. 2014, [Online] Available: http://arxiv.org/abs/1412.6564.

[20] MoGo 2015. https://senseis.xmp.net/?MoGo, date retrieved: April 13, (2021).

[21] X. Niu and M. Müller. “An open boundary safety-of-territory solver for the game of Go,” J. van den Herik, P. Ciancarini, and H. Donkers, editors, Computers and Games, volume 4630 of Lecture Notes in Computer Science, 37–49, Torino, Italy, Springer, (2007).

[22] M. Müller. (1997). “Playing it safe: Recognizing secure territories in computer Go by using static rules and search,” H. Matsubara, editor, Game Programming Workshop in Japan ’97, pages 80–86, Computer Shogi Association, Tokyo, Japan.

[23] M. Müller. (1999). “Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames,” Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), 578–583, Stockholm, Sweden.

[24] M. Müller. (2001). “Review: Computer Go 1984 - 2000,” T. Marsland and I. Frank, editors, Computers and Games 2000, number 2063 in Lecture Notes in Computer Science, 426–435, Springer Verlag.

[25] M. Müller. (2002). “Computer Go”, Artificial Intelligence, 134(1–2):145–179.

[26] X. Niu and M. Müller. (2008). “An improved safety solver in Go using partial regions,” J. van den Herik, X. Xu, Z. Ma, and M.Winands, editors, Computers and Games, volume 5131 of Lecture Notes in Computer Science, 102–112, Beijing, China, Springer.

[27] X. Niu, A. Kishimoto, and M. Müller. (2006). “Recognizing seki in computer Go,” J. van den Herik, S.-C. Hsu, T.-s. Hsu, and H. Donkers, editors, Advances in Computer Games, volume 4250 of Lecture Notes in Computer Science, 88–103, Springer.

[28] Sarkar, P. (2000). “A brief history of cellular automata,” ACM Computing Surveys, 32(1):80–107.

[29] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. (January 2016). “Mastering the game of Go with deep neural networks and tree search,” Nature, 529, 484–489.

[30] David Silver, et al. (2018). “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” Science 362(6419):1140–1144.

[31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche. (2017). Thore Graepel and Demis Hassabis, “Mastering the game of Go without human knowledge,” Nature, 550.

[32] T. Thomsen. (2000). “Lambda-search in game trees with application to Go,” ICGA Journal, 23(4):203–217.

[33] Jonathan K. Millen. (April 1982). “Programming the Game of Go,” BYTE, 6:102–121.

[34] Walter Reitman and Bruce Wilcox. (August 1979). “The structure and performance of the INTERIM.2 Go program,” IJCAI’79: Proceedings of the 6th international joint conference on Artificial intelligence, 2:711–719.

[35] B. Wilcox. (1988). “Computer Go,” D. Levy, editor, Computer Games, 2, 94–135, Springer-Verlag.

[36] T. Wolf. (2000). “Forward pruning and other heuristic search techniques in tsume go,” Information Sciences, 122:59–76.

[37] Von Neumann John. (1966). “The Theory of Self-Reproducing Automata,” A. W. Burks, ed., University of Illinois Press, Urbana.

[38] Zen (Go Program) 2018. https://senseis.xmp.net/?ZenGoProgram, date retrieved: April 13, (2021).

[39] Albert L. Zobrist. (1990). “A New Hashing Method with Application for Game Playing,” ICGA Journal,13:69–73.