Hex game winner prediction by the use of Tsetlin Machine

by

Yusef Said & Harald Borgenvik

in

IKT457 Learning Systems

Supervised by Vojtech Halenka

Department of Information and Communication Technology

University of Agder

Grimstad, December 2024

Contents

1 Introduction

Hex is a board game that Piet Hein invented in 1942 [1]. It is a two-player game, where the goal of each player is to create a connection from one side to the other. One of the interesting properties of the game is that it can't end in a draw. It has gained the interest of mathematicians and researchers in related areas such as computer and data science [2]. Figure 1.1 displays a typical end-game position, where the blue player wins and the red player loses.

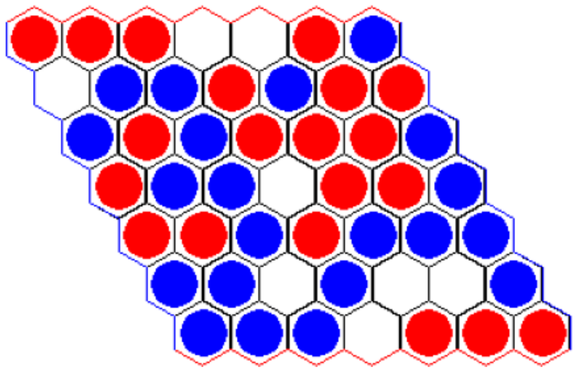


Figure 1.1: A game that has been won by the blue player. The goal of the game can be said to connect the first and last column, and the first and last row for the blue and red players, respectively. The game is from the Kaggle dataset of completed games [3].

The objective of this project is to predict the winner at specific points in the game using a Tsetlin Machine (TM):

- At the end of the game.
- Two moves before the end.
- Five moves before the end.

For this project the dataset from Kaggle.com will be utilized [3]. A limitation of the dataset is that it only contains the final position of each game and, not the history of moves throughout the game. For the remaining two points, two and five moves before the end, different datasets have been used. Later in this report, a solution to produce such data will be explained in detail in section 2.5.2. An attempt at analyzing some of the clauses learned by the TM will be done in 4. The project source code and additional resources are available on GitHub at: `https://github.com/haraldbo/ikt-457-hex-game-winner-prediction`.

1.1 Overview of the Tsetlin Machine

The TM was invented by professor Ole-Christoffer Granmo and published in 2018 [4] and is based on the concept of learning automata, discovered by Mikhail Lvovitsj Tsetlin. TM has gained interest as an alternative to artificial neural networks. It is based on simple boolean clauses, and this makes it both easy to interpret and potentially very efficient [5]. Since its discovery, TM has been applied to several different areas such as natural language processing [6], nonlinear regression [7], pattern recognition in images [8] and games like Chess [9] and Hex [10], [11].

2 Method: towards a solution

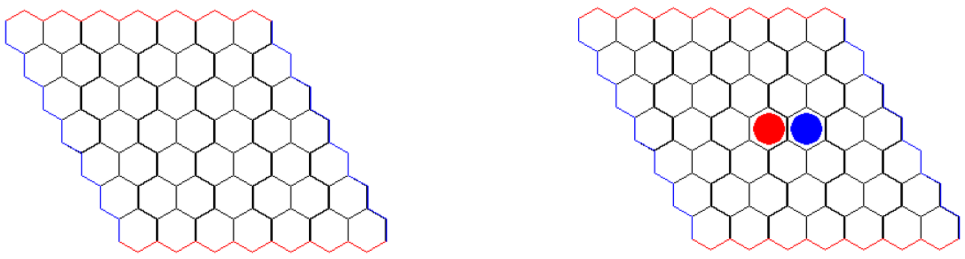
This chapter contains ideas investigated and used in the proposed solutions for the hex game project. The proposed solutions can be found in section 2.6.

2.1 The Game

Hex boards are mostly rhombus-shaped as visually displayed in figure 2.1, and come in different sizes.

Figure 2.1: Visual display of two typical rhombus-shapes.

The board in figure 2.2 is a 7x7 board, 7 columns wide and 7 rows tall. Other common board sizes are 9x9, 11x11, and 13x13 [12].



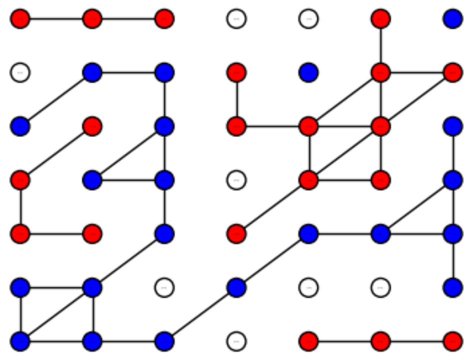
(a) Empty hex game board. **(b)** Both players, red and blue, have made a move.

Figure 2.2: Start of the game.

The hex game board starts empty (Figure 2.2a). The starting player then occupies a cell on the board, for example, the blue player occupies the cell at position $(x = 5, y = 4)$, and can be shortened to be cell (5, 4). After blue has made a move, it is the red player's turn to occupy a cell, for example, $(4, 4)$. The board then looks like figure 2.2b. The game continues in this alternating way until one of the players has created a connection across the board. The first player to move has an advantage, and one way to mitigate this is by the use of *the swap rule*. The swap rule allows the second player to switch colors and thereby acquire ownership of the hexagon that was captured in the first move [2].

2.2 Representations of the board and the game

Since the goal of the hex game is to create a connection from one side of the board to the other, like the road leading from one city to the other, the graph data structure comes easily to mind. Each hexagonal cell on the board, 49 in total for a 7x7 board, can be considered nodes, and when two hexagons of the same color are placed side by side, they can be connected by an edge. With this approach, the board in figure 1.1 can be represented as a graph (figure 2.3).


(a) Graph representation of the board in figure 2.3b

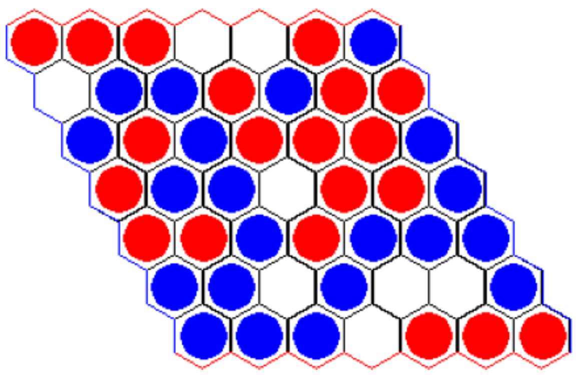

(b) A hex game example.

Figure 2.3: Visual comparison of the same board where the board can be represented as a graph.

One thing to note is that the number of possible connections of each cell differs depending on the location on the board. There are four corners in the rhombus-shaped board. The cell in the upper left corner at (0, 0) and the lower right corner at (6, 6) can only be connected to two other cells, while the cell located in the upper right corner at (6, 0) and the cell located lower left corner at (0, 6) can be connected to three neighboring cells. A visual representation of a 7x7 hex board is displayed in Figure 2.4 and the coordinates for each cell.

The cells that have two neighboring cells are displayed in figure 2.5:

- (0, 0) can be connected to (0, 1) and (1, 0) as displayed in figure 2.5a.
- (6, 6) can be connected to (6, 5) and (5, 6) as displayed in figure 2.5b.

The cells that have three neighboring cells are displayed in figure 2.6:

- (6, 0) can be connected to (6, 1), (5, 1) and (5, 0) as displayed in figure 2.6a.
- (0, 6) can be connected to (0, 5), (1, 5) and (1, 6) as displayed in figure 2.6b.

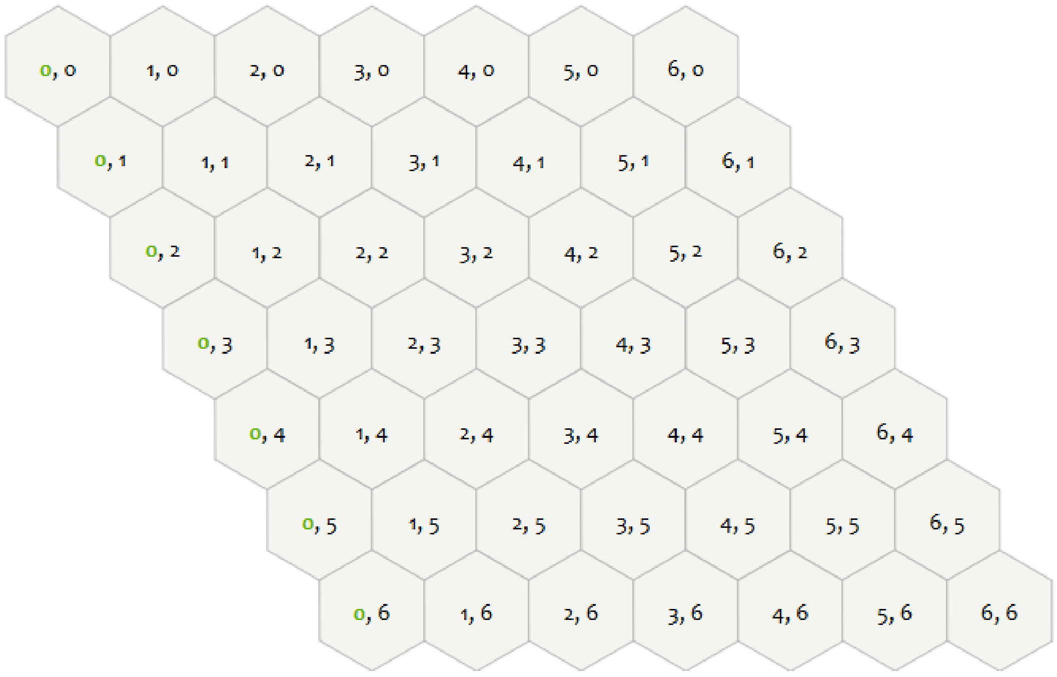
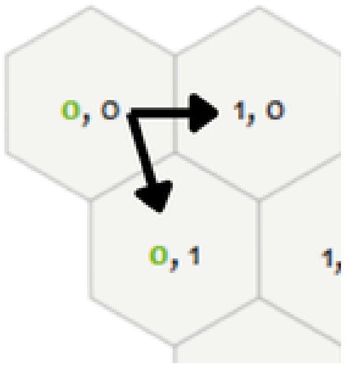
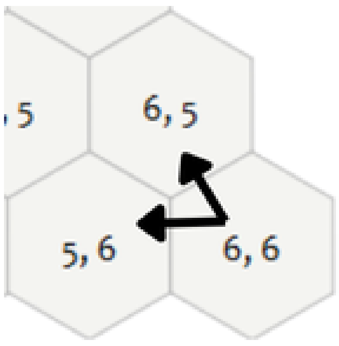


Figure 2.4: Visual representation of a rhombus hex game, with their respective numbered cells [13].



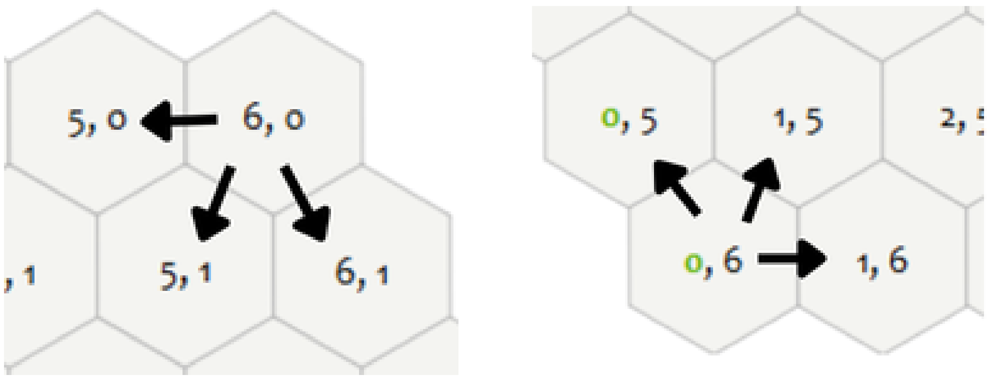
(a) Cell (0, 0) and its neighbors.



(b) Cell (6, 6) and its neighbors.

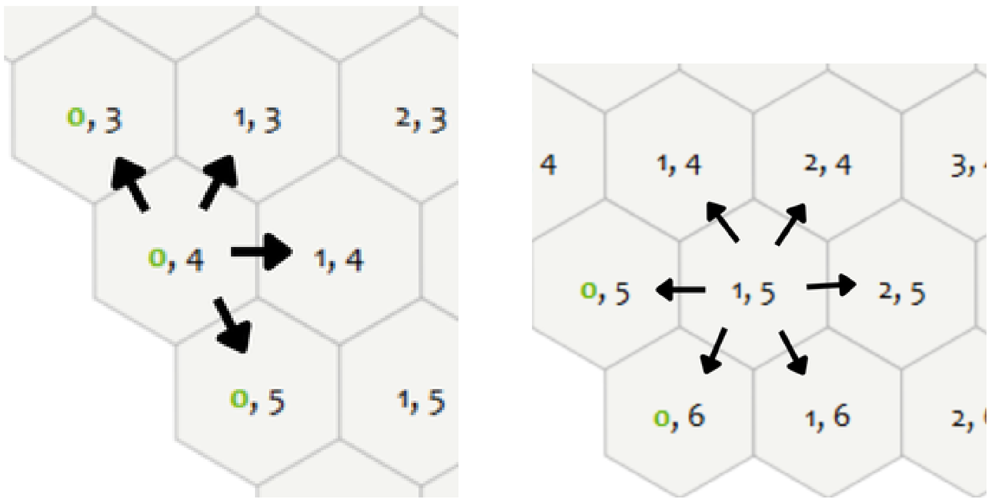
Figure 2.5: Illustration of two corner cells, (0,0) and (6,6), showing the two possible connections to neighboring cells. The arrows indicate the direction and specific neighboring cells each corner cell can connect to [13].

The cells positioned at the edge of the board have four neighboring cells (Figure 2.7a). The cells that are not positioned at the edge have six neighboring cells (Figure 2.7b).



(a) Cell (6, 0) and its neighbors. **(b)** Cell (0, 6) and its neighbors.

Figure 2.6: Illustration of two corner cells, (6,0) and (0,6), showing the three possible connections to neighboring cells. The arrows indicate the specific neighboring cells each corner cell can connect to [13].



(a) Cell (0, 4) and its four neighbors. **(b)** Cell (1, 5) and its six neighbors.

Figure 2.7: Illustration of two cells on the board (0,4) and (1, 5), showing the four and six possible connections to neighboring cells. The arrows indicate the neighboring cells it can connect to [13].

2.3 Winning conditions

When trying to predict the winner of a game, the many ways to win a game are of great interest. As displayed in figure 2.3, the blue player has constructed a path from the cell (0,6) to (6, 4) and thereby won the game. If three functions p, b and r are defined such that:

$$p(x_0, y_0, x_1, y_1) = \begin{cases} 1, & \text{when there is a path between } (x_0, y_0) \text{ and } (x_1, y_1). \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

$$b(x,y) = \begin{cases} 1, & \text{when player at } (x,y) \text{ is blue.} \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

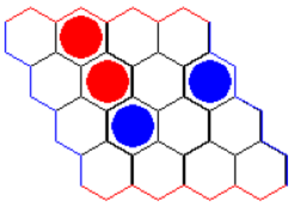
$$r(x,y) = \begin{cases} 1, & \text{when player at } (x,y) \text{ is red.} \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

Then the winning condition for the blue and red player on a 7x7 board can be formulated as in the equations 2.4 and 2.5.

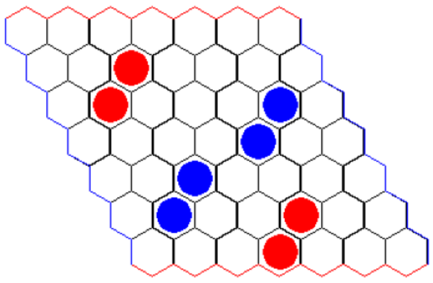
$$\text{Blue wins when } \sum_{i=0}^{6}\sum_{j=0}^{6} p(0,i,6,j)b(0,i)b(6,j) > 0 \tag{2.4}$$

$$\text{Red wins when } \sum_{i=0}^{6}\sum_{j=0}^{6} p(i,0,j,6)r(i,0)r(j,6) > 0 \tag{2.5}$$

It has been mathematically proven that there exists a winning strategy for the first player [14], and algorithms for solving some board sizes such as 7x7 and 8x8 in a reasonable amount of time have been developed [15][16]. In some board configurations, it is easy to tell the winning player, such as in Figure 2.8. This common pattern is known as *bridge* [17].



(a) The red player can not stop the blue player from forming a connection and winning the game.



(b) The same pattern as 2.8a. Two empty cells are between each blue cell, and the blue player is unstoppable.

Figure 2.8: Example of a common pattern

Other common patterns include the wheel, the trapezoid, the crescent, and several others [18].

2.4 Feature extraction and booleanization

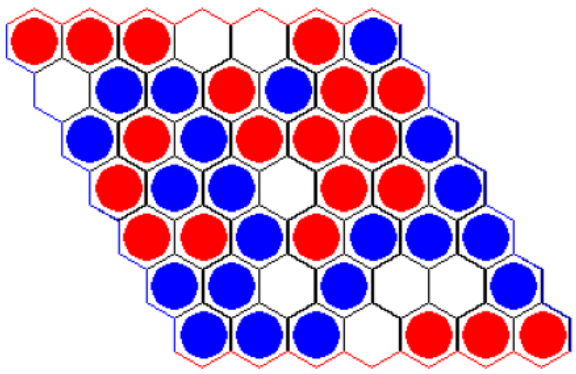
Before one can create a TM-based solution, one must decide on what features to use, and how they can be booleanized. Boolanization means reducing features to boolean values such as True and False. The hex board in figure 1.1 can be represented as a 2D array, like in figure 2.9. The red player (-1) has occupied the cell in the upper left corner. The blue player (1) has occupied the cell in the upper right corner. The cells containing a 0 are unoccupied.

If one is to booleanize the occupied and available cells on the board, then a problem arises. Each cell on the board can hold three values (-1, 0, and 1), one too many for booleanization.

-1	-1	-1	0	0	-1	1
0	1	1	-1	1	-1	-1
1	-1	1	-1	-1	-1	1
-1	1	1	0	-1	-1	1
-1	-1	1	-1	1	1	1
1	1	0	1	0	0	1
1	1	1	0	-1	-1	-1

-1	-1	-1	0	0	-1	1
0	1	1	-1	1	-1	-1
1	-1	1	-1	-1	-1	1
-1	1	1	0	-1	-1	1
-1	-1	1	-1	1	1	1
1	1	0	1	0	0	1
1	1	1	0	-1	-1	-1

(a) 2D array representation of the board in figure 2.9b



(b) A hex game example.

Figure 2.9: Visual comparison of the same board where the board can be represented as a 2D array.

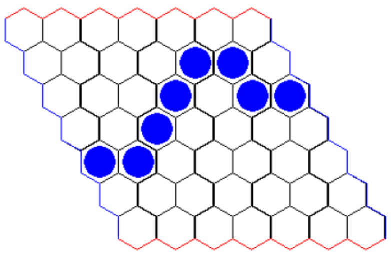
In Haddeland and Simonsens thesis, a simple solution to the booleanization problem was proposed [10]. It involves expanding the board and splitting it into a left and a right side. The left side of the board is concerned with cells occupied by the blue player, the right side is for the red player. When applying this approach to the board in 2.9, one gets the board in figure 2.10, a new 2D array where each entry is a value of either 0 or 1, which is booleanizable.

0	0	0	0	0	0	1	1	1	1	0	0	1	0
0	1	1	0	1	0	0	0	0	0	1	0	1	1
1	0	1	0	0	0	1	0	1	0	1	1	1	0
0	1	1	0	0	0	1	1	0	0	0	1	1	0
0	0	1	0	1	1	1	1	1	0	1	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	1	1	1

Figure 2.10: Expanded 2D array representation of the board in figure 2.9. Every entry holds a value of 0 or 1, and can therefore easily be booleanized.

The board in itself can be sent to the TM as a list of what positions are occupied, and by whom, but would this work well? Wouldn't the TM then have to memorize all possible ways the red and blue player can win? There are numerous ways to win for each player. The blue player can connect one side to the other side in more ways than can be stored in computer memory.

As mentioned in section 2.2, the board can be represented as a graph. The player is interested in making a connection from one side to the other, and the graph structure is well-suited for this kind of task. Graphs are often used to find the shortest path, or the existence of a path, between nodes. Since connections are so essential, an interesting question then arises: How can connections in the graph be booleanized?


(a) Connection 1.

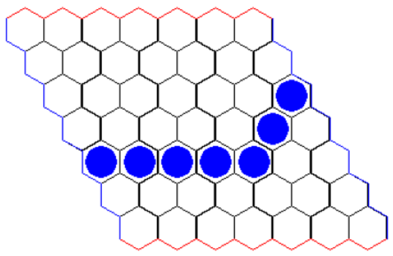

(b) Connection 2.

Figure 2.11: Two blue connections between cell $(0, 4)$ to $(6, 2)$.

A simple way to booleanize anything is by asking a yes-no question; Is the cell $(0, 0)$ occupied by red? Is the number of cells on the board an even number? Has the blue player created a connection between $(0, 4)$ and $(6, 2)$? Is Blue the starting player? From Figure 2.11, In figure 2.11a and 2.11b, one can observe that the blue player has created a connection from $(0, 4)$ to $(6, 2)$. There are numerous ways that the blue player can construct such a connection, but it may not be important how it is done, only that there exist a connection. The question "**Has the blue player created a connection between (0, 4) and (6,2)?**" holds all possible connections from $(0, 4)$ to $(6, 2)$ in only one question. This is one way to booleanize connections. Has the blue or red player connected cell `n` with cell `m`? Inspired by the way the board was divided into two in Haddeland and Simonsens thesis [10], one can ask the connectivity questions for each of the players.

- Has blue player connected $(0, 0)$ and $(6, 0)$?
- Has red player connected $(0, 0)$ and $(6, 0)$?
- Has blue player connected $(0, 0)$ and $(6, 1)$?
- Has blue red connected $(0, 0)$ and $(6, 1)$?
- and so on...

If one were to ask if there is a connection between cell n and cell m for all n and m cells on the board, how many questions would one have to ask? The blue player can connect $(0, 0)$ to $(1, 0)$, $(2, 0)$, ..., $(6, 6)$, so there are 48 possible connection questions from $(0, 0)$ to the other cells. The question of whether $(1,0)$ is connected to $(0,0)$ has already been asked, but one can ask if it has been connected to $(2,0)$, $(3,0)$,...,$(6,6)$, so there are 47 possible questions from $(1,0)$ to the other cells. From $(2,0)$ there are 46 possible questions, and so on. For a 7x7 board, the number of questions adds up to

$$\sum_{i=1}^{48} i = \frac{48(48+1)}{2} = 1176 \text{ possible connection questions for blue player}$$

For both players, there are $1176 \cdot 2 = 2352$ possible questions. For an NxN board, there are $\frac{N^2(N^2-1)}{2}$ connection questions for each of the players, which means that there are $N^2(N^2-1)$ in total.

2.5 Datasets

2.5.1 Completed games

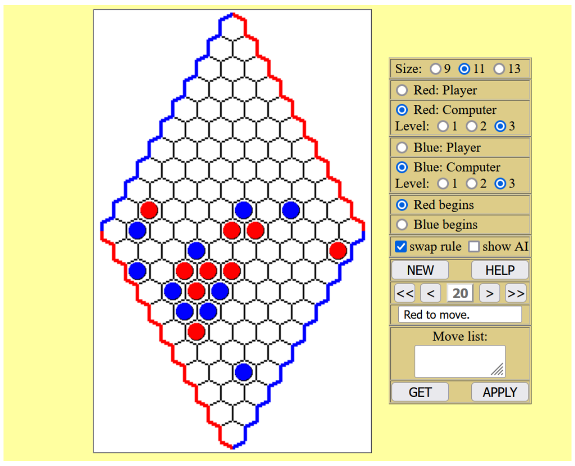
The dataset that was used for completed games was provided by Charles Hollingsworth on kaggle.com [3]. It consists of one million 7x7 games that have ended in one of the players reaching the goal of making the decisive connection. The dataset is contained in a `.csv` file where each row represents the board configuration and the winner. The first 49 columns contain information about which of the players have occupied the cells on the board, or if it is unoccupied. The last column states who has won the game.

The first 100, 000 rows of the dataset have been used. It has been split into an 80/20 stratified split for training and testing respectively. The dataset is slightly unbalanced; about 46% of the games are won by red, whereas 54% are won by blue.

2.5.2 Two and five moves before the end

The dataset for two and five moves before the end of the game was not provided. The definition of two and five moves before the end was not clearly defined, and this was interpreted as freedom to experiment and choose.

The datasets that have been utilized for these goals have been obtained by observing and recording two AIs playing against each other on lutanho.net [12] as displayed in Figure 2.12. A Python script that records games by interpreting pixels on screen was developed for this purpose. The script captures the screen many times each second and checks for new moves. When a game is won by one of the players, the move history is appended to a `.csv` file. By saving all moves done by the AI, one can obtain datasets for board configurations at two and five moves before the end by not including the two and five last moves, respectively (See figure 2.13).


Size: 9 [✓ 11] 13

Red: Player

[✓ Red: Computer]

Level: 1 2 [✓ 3]

Blue: Player

[✓ Blue: Computer]

Level: 1 2 [✓ 3]

[✓ Red begins]

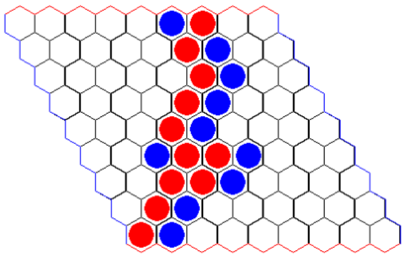
Blue begins

[✓ swap rule] show AI

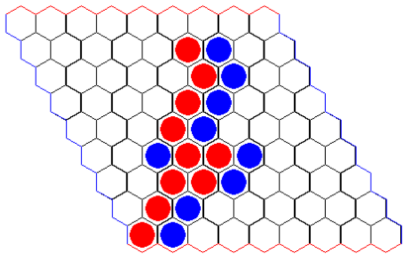
Red to move.

Move list:

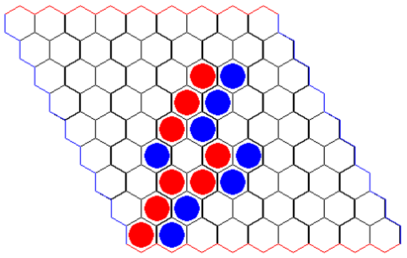
Figure 2.12: Getting two AIs to play against each other.



(a) Ended in red winning



(b) Red wins in two moves.



(c) Red wins in five moves.

Figure 2.13: Board configurations from a captured game where red wins. Red was the starting player in this game.

One issue with this approach is that the AI bots on lutanho.net tend to repetitively play the same games over and over. A measure to remedy this was implemented by randomly toggling the radio buttons for each of the bot player levels after each round had ended. This resulted in more varied games, and a total of 5157 and 5035 unique board configurations were captured for two and five moves before the end, respectively. The script had to run for a few days to gather all the data. A flowchart representing how the datasets were obtained is illustrated in figure 2.14.

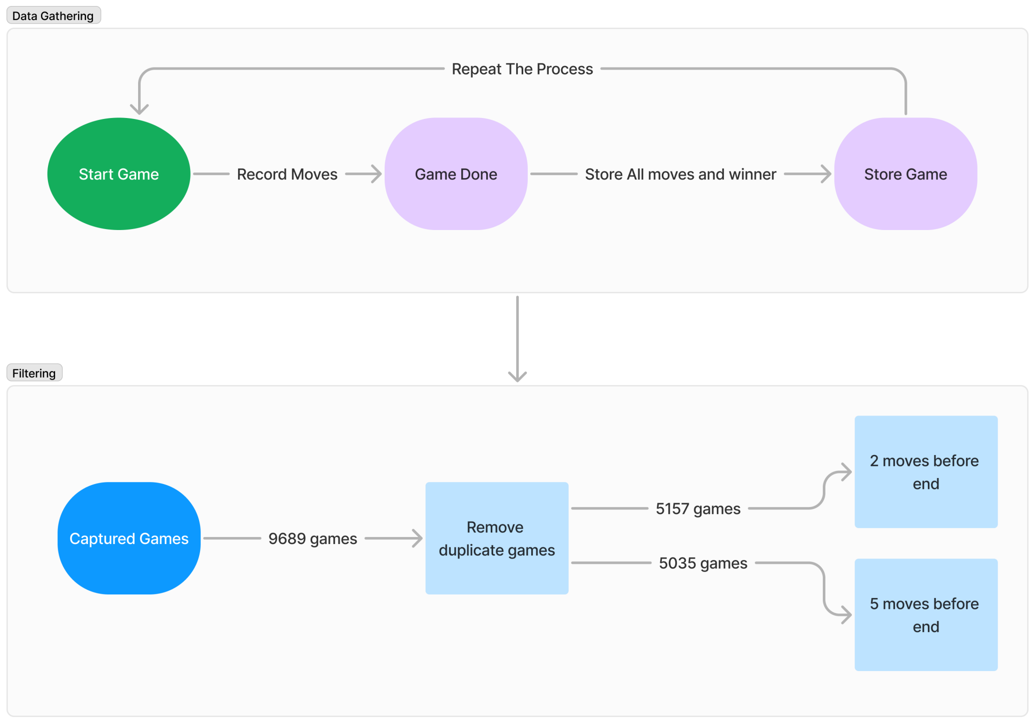


Figure 2.14: Process of gathering, filtering,g and constructing the two and five moves before the end dataset

An issue with this approach is that when going two or five moves back in history, it may be an error to state that the winner is the player who won the game. The win of the player may be attributed to a blunder by the other player that was made in the last two or five moves. This is however what was conducted in this project, and the performance reflects this decision.

2.6 Proposed solution

The proposed solution is meant to work well for all the three goals mentioned in the introduction: end of the game, two moves before the end, and five moves before the end. Finding a solution that predicts the winner at the end of the game is very simple because then one only needs to check if there is a connection from the cells in the first to the last row (red player), or from the cells in the leftmost to the rightmost column (blue player). Having to predict the winner before the game is over is a more challenging task.

The crux of the game is to connect one side to the other, while at the same time preventing the other player from doing the same. The proposed solution is one based on occupied cells and connections. The symbols used can be viewed in table 2.1.

Symbol	**Description**	**Count**
$CB_{(x_0,y_0),(x_1,y_1)}$	The symbols starting with CB are set to true when blue has made a connection from (x_0, y_0) to (x_1, y_1).	$\frac{N^2(N^2-1)}{2}$
$CR_{(x0,y0),(x1,y1)}$	The symbols starting with CR are set to true when red has made a connection from (x_0, y_0) to (x_1, y_1).	$\frac{N^2(N^2-1)}{2}$
$R_{x,y}$	$R_{x,y}$ is set to true when red has occupied cell (x, y)	N^2
$B_{x,y}$	$B_{x,y}$ is set to true when blue has occupied cell (x, y)	N^2
All		$N^2(N^2 + 1)$

Table 2.1: Description of symbols. N is the size of a NxN board. This means that for a 7x7 board, there are a total of 2450 symbols.

When none of the players have occupied a cell, for example, cell $(2, 1)$, then both $R_{2,1}$ and $B_{2,1}$ will be set to false, this implies that the cell is unoccupied. Unoccupied cells can be occupied in the next moves, and some unoccupied cells are more attractive than others. The assumption is that unoccupied cells are essential when one wants to teach the TM to detect patterns such as the bridge pattern (see figure 2.8).

One possible improvement to reduce the number of symbols is that the winning connections, from the top to bottom row for the red player, and left to right column for the blue player, can be represented by 2 symbols instead of $2N^2$ number of connection symbols. This is because all connections from top to bottom carry the same information: **that the red player has won**. It is the same for the connections from the leftmost and rightmost columns and the blue player. This has not been implemented.

The starting player is an important feature that has not been included in the proposed solution. For the dataset that has been utilized for two and five moves before the end, the red player is the starting player in all of the games. If one uses datasets where the starting player varies, adding the starting player as a symbol seems essential. An alternative to adding the property may be to invert the colors of the pieces on the board and transpose the board matrix (flipping the board matrix across the diagonal). With this idea, a way of reducing the number of needed symbols may be obtained through experimentation.

Before training can start, symbols must be extracted from the board configuration. To booleanize the symbols prefixed by CB and CR, the Python library Networkx [19] has been utilized for graph construction and to check for the existence of connections between occupied cells of the same color. An overview of the process can be viewed in figure 2.15.

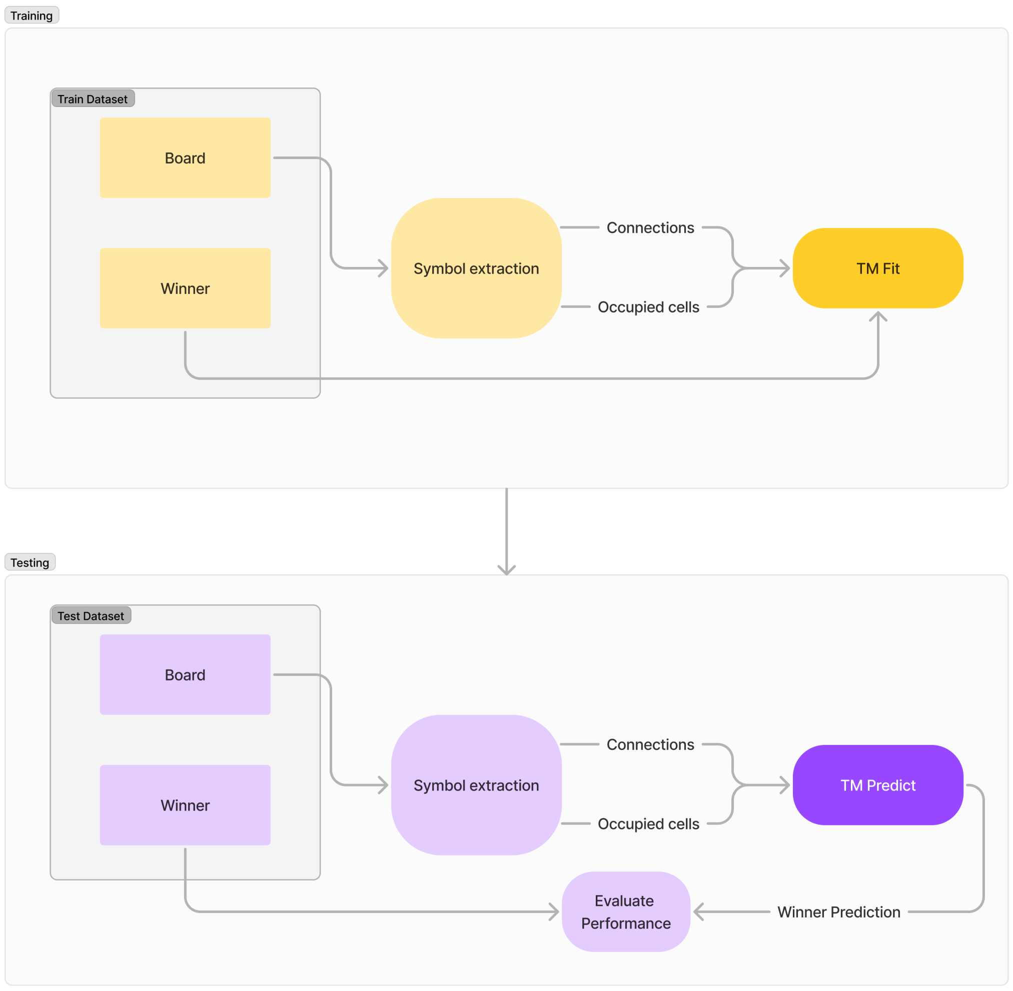


Figure 2.15: Training and test flow

The Graph Tsetlin Machine library [20] with cuda support has been utilized to train and test the solution. Only one cell has been used in this proposed solution, so it can be said to be a vanilla TM [20].

3 Results and Discussions

3.1 Completed games

The proposed TM-based solution achieves almost 100% accuracy when predicting the winner on the 7x7 Kaggle dataset with completed games. A search for optimal hyperparameters has not been carried out, and it is likely possible to find hyperparameters that will result in improved performance. The hyperparameters used can be viewed in Table 3.1. The maximum accuracy achieved for varying numbers of clauses can be viewed in Figure 3.1.

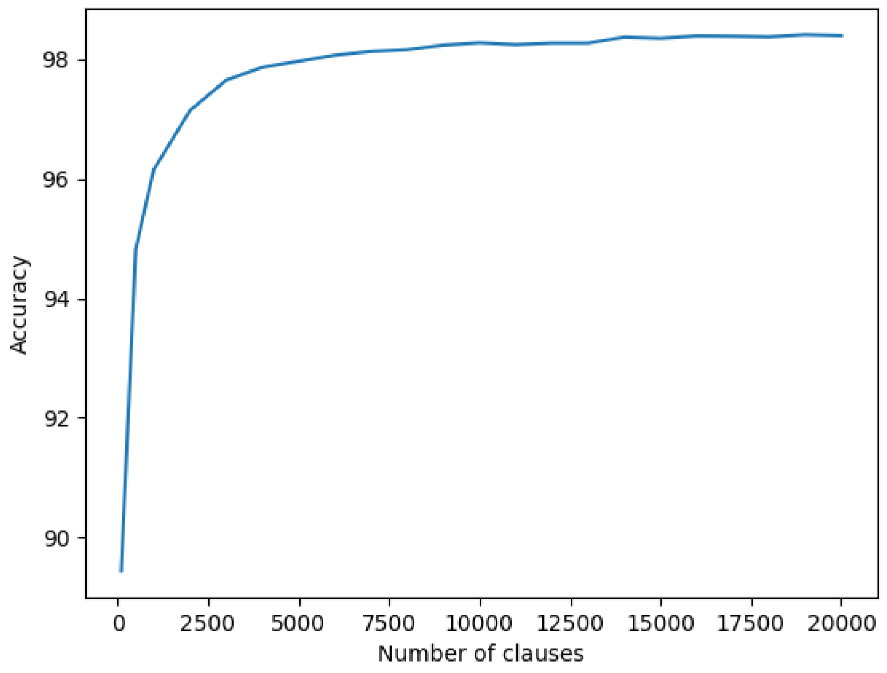


Figure 3.1: Max accuracy achieved for different numbers of clauses.

As mentioned, there are N^2 winning connections for each of the players. For a 7x7 board, there are $2 \cdot 7^2 = 98$ such decisive connections. One might then think that 100 clauses should be sufficient to reach 100% accuracy, but only 88% was achieved. The accuracy increases steeply between 100 and 2000 clauses, and after that the gain is minuscule.

Number of clauses	Varying from 100 to 20000
S	17
T	20 000
Hypervector size	512
Hypervector bits	2
Message size	256
Message bits	1
State bits	16

Table 3.1: Hyperparameters used during training and testing on the dataset of completed games

The reason why it is unable to reach 100% may be because it has not yet seen and learned all the 98 winning connections. 100,000 rows from the Kaggle dataset were used and it was split into training and testing. The training dataset contains 80,000 rows and the testing dataset contains 20,000 rows. There may be some boards in the testing dataset with winning connections that are not present in the training dataset? This may be considered an argument in favor of combining these winning connections into two symbols instead of 98. The highest accuracy achieved during this experiment was 98.4% at 20,000 clauses. Utilizing a larger portion of the Kaggle dataset may increase the accuracy.

3.2 Two moves before the end

The proposed solution performed well on the two moves before the end dataset. The highest achieved accuracy was 98.75%. A light grid search for the best combination of s and number of clauses was carried out with number of clauses varying from 2000 to 20,000 and s values in range from 10 to 28 (see Figure 3.2). All combinations reached above 95% accuracy. The highest accuracy was achieved with 18,000 clauses and $S = 18$. The scores were very even and it may be a coincidence that this combination scored the highest.

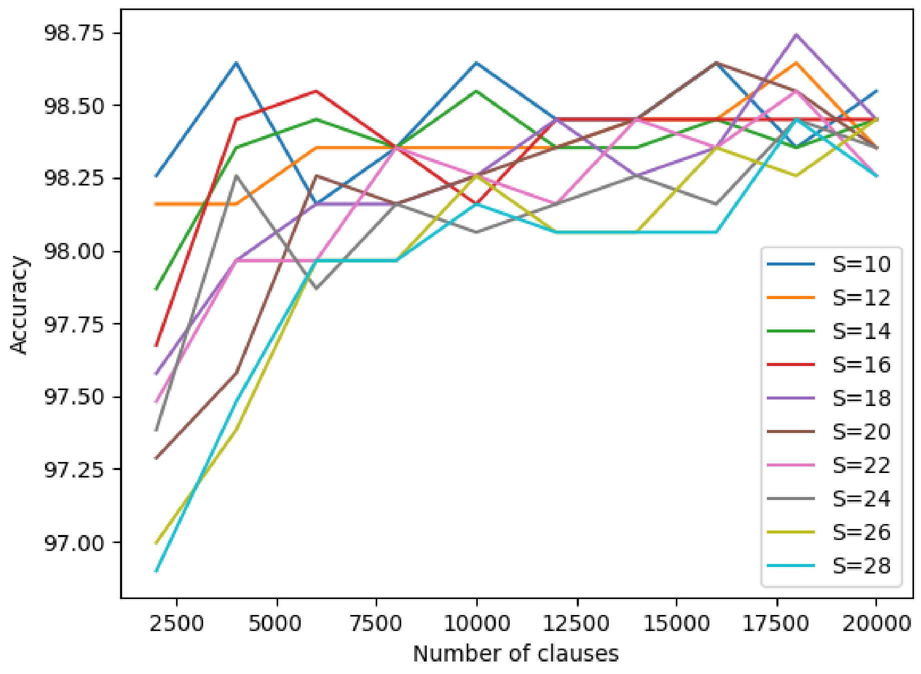


Figure 3.2: Highest accuracy achieved with different combinations of number of clauses and s values.

Even a low number of clauses, such as 2000, did perform well, the hyperparameters that were utilized, can be viewed in Table 3.2.

Number of clauses	Varying from 100 to 20000
S	Varying from 10 to 28
T	20000
Hypervector size	2048
Hypervector bits	1
Message size	256
Message bits	1
State bits	16

Table 3.2: Hyperparameters used during training and testing on the two moves before the end dataset.

3.3 Five moves before the end

The proposed solution performed well on the five moves before the end dataset. The highest accuracy (95.1%) was achieved with 10,000 clauses and $S = 20$. An accuracy of 92.8% was achieved by utilizing only 1000 clauses and $S = 10$. The hyperparameter configurations that were used can be viewed in Table 3.3. Figure 3.3 displays the maximum accuracy that was achieved by varying numbers of clauses and specificity.

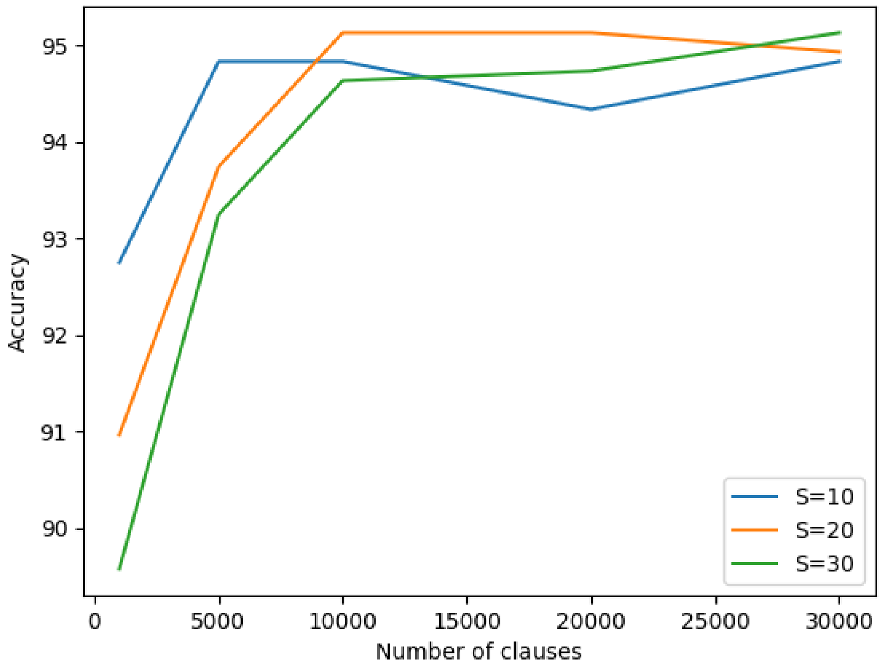


Figure 3.3: Highest accuracy achieved with different combinations of number of clauses and s values on the five moves before the end dataset

Number of clauses	1000, 5000, 10000, 20000, 30000
S	10, 20, 30
T	20000
Hypervector size	1024
Hypervector bits	2
Message size	256
Message bits	2
State bits	16

Table 3.3: Hyperparameters used during training and testing on the five moves before the end dataset

4 Interpretability analysis

Interpretability is considered one of the advantages of TM compared to other machine learning paradigms. The clauses that are outputted by the TM are usually easy to interpret and evaluate. In this chapter, an attempt will be made at analyzing some of the clauses that have been learned by the machine.

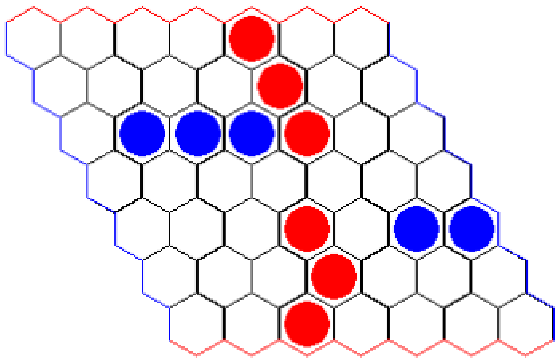


Figure 4.1: Blue cannot stop red from winning.

It is expected that the proposed TM-based solution will be able to learn winning patterns and that this will be reflected by the learned clauses. A winning board configuration for the red player is depicted in Figure 4.1. This is a common pattern where there is no way for the blue player to stop the red player from forming a winning connection. What makes this a winning connection can be summarized in four points:

- Red has formed a connection from $(4, 0)$ to $(4, 2)$
- Red has formed a connection from $(3, 4)$ to $(2, 6)$
- Cell $(3, 3)$ is not occupied by blue.
- Cell $(4, 3)$ is not occupied by blue.

A winning clause may then be encoded as such:

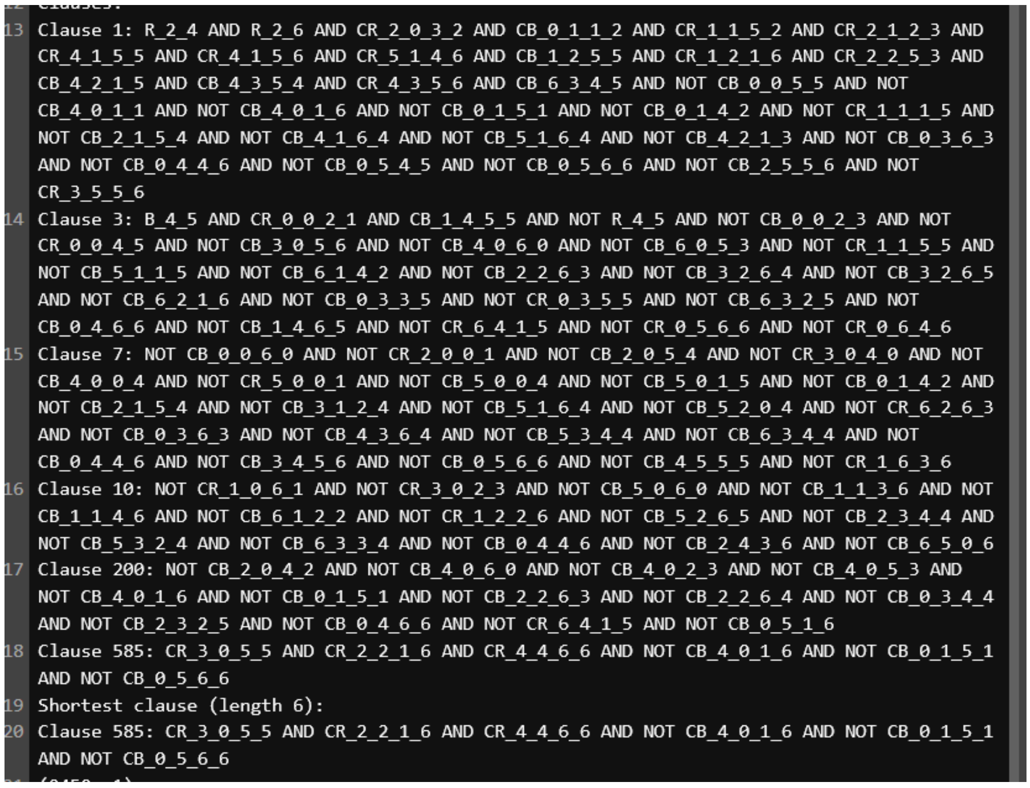
$$\text{Red}_{\text{Win}} = RC_{(4,0),(4,2)} \wedge RC_{(3,4),(2,6)} \wedge \neg B_{3,3} \wedge \neg B_{4,3}$$

The goal of this chapter is to have a closer examination of some generated clauses and evaluate if some such patterns can be recognized.

4.1 Clause analysis

4.1.1 Example 1

Code for viewing clauses has been implemented, and some of the learned clauses can be seen in Figure 4.2 displaying the terminal output. These clauses represent how the TM interprets the hex game board for each player by considering occupied cells and connections between them.



```
13 Clause 1: R_2_4 AND R_2_6 AND CR_2_0_3_2 AND CB_0_1_1_2 AND CR_1_1_5_2 AND CR_2_1_2_3 AND
   CR_4_1_5_5 AND CR_4_1_5_6 AND CR_5_1_4_6 AND CB_1_2_5_5 AND CR_1_2_1_6 AND CR_2_2_5_3 AND
   CB_4_2_1_5 AND CB_4_3_5_4 AND CR_4_3_5_6 AND CB_6_3_4_5 AND NOT CB_0_0_5_5 AND NOT
   CB_4_0_1_1 AND NOT CB_4_0_1_6 AND NOT CB_0_1_5_1 AND NOT CB_0_1_4_2 AND NOT CR_1_1_1_5 AND
   NOT CB_2_1_5_4 AND NOT CB_4_1_6_4 AND NOT CB_5_1_6_4 AND NOT CB_4_2_1_3 AND NOT CB_0_3_6_3
   AND NOT CB_0_4_4_6 AND NOT CB_0_5_4_5 AND NOT CB_0_5_6_6 AND NOT CB_2_5_5_6 AND NOT
   CR_3_5_5_6
14 Clause 3: B_4_5 AND CR_0_0_2_1 AND CB_1_4_5_5 AND NOT R_4_5 AND NOT CB_0_0_2_3 AND NOT
   CR_0_0_4_5 AND NOT CB_3_0_5_6 AND NOT CB_4_0_6_0 AND NOT CB_6_0_5_3 AND NOT CR_1_1_5_5 AND
   NOT CB_5_1_1_5 AND NOT CB_6_1_4_2 AND NOT CB_2_2_6_3 AND NOT CB_3_2_6_4 AND NOT CB_3_2_6_5
   AND NOT CB_6_2_1_6 AND NOT CB_0_3_3_5 AND NOT CR_0_3_5_5 AND NOT CB_6_3_2_5 AND NOT
   CB_0_4_6_6 AND NOT CB_1_4_6_5 AND NOT CR_6_4_1_5 AND NOT CR_0_5_6_6 AND NOT CR_0_6_4_6
15 Clause 7: NOT CB_0_0_6_0 AND NOT CR_2_0_0_1 AND NOT CB_2_0_5_4 AND NOT CR_3_0_4_0 AND NOT
   CB_4_0_0_4 AND NOT CR_5_0_0_1 AND NOT CB_5_0_0_4 AND NOT CB_5_0_1_5 AND NOT CB_0_1_4_2 AND
   NOT CB_2_1_5_4 AND NOT CB_3_1_2_4 AND NOT CB_5_1_6_4 AND NOT CB_5_2_0_4 AND NOT CR_6_2_6_3
   AND NOT CB_0_3_6_3 AND NOT CB_4_3_6_4 AND NOT CB_5_3_4_4 AND NOT CB_6_3_4_4 AND NOT
   CB_0_4_4_6 AND NOT CB_3_4_5_6 AND NOT CB_0_5_6_6 AND NOT CB_4_5_5_5 AND NOT CR_1_6_3_6
16 Clause 10: NOT CR_1_0_6_1 AND NOT CR_3_0_2_3 AND NOT CB_5_0_6_0 AND NOT CB_1_1_3_6 AND NOT
   CB_1_1_4_6 AND NOT CB_6_1_2_2 AND NOT CR_1_2_2_6 AND NOT CB_5_2_6_5 AND NOT CB_2_3_4_4 AND
   NOT CB_5_3_2_4 AND NOT CB_6_3_3_4 AND NOT CB_0_4_4_6 AND NOT CB_2_4_3_6 AND NOT CB_6_5_0_6
17 Clause 200: NOT CB_2_0_4_2 AND NOT CB_4_0_6_0 AND NOT CB_4_0_2_3 AND NOT CB_4_0_5_3 AND
   NOT CB_4_0_1_6 AND NOT CB_0_1_5_1 AND NOT CB_2_2_6_3 AND NOT CB_2_2_6_4 AND NOT CB_0_3_4_4
   AND NOT CB_2_3_2_5 AND NOT CB_0_4_6_6 AND NOT CR_6_4_1_5 AND NOT CB_0_5_1_6
18 Clause 585: CR_3_0_5_5 AND CR_2_2_1_6 AND CR_4_4_6_6 AND NOT CB_4_0_1_6 AND NOT CB_0_1_5_1
   AND NOT CB_0_5_6_6
19 Shortest clause (length 6):
20 Clause 585: CR_3_0_5_5 AND CR_2_2_1_6 AND CR_4_4_6_6 AND NOT CB_4_0_1_6 AND NOT CB_0_1_5_1
   AND NOT CB_0_5_6_6
```

Figure 4.2: Terminal displaying some of the clauses that vote in favor of the red player winning.

One of the output clauses will be examined to better understand the TM's reasoning. From the terminal output 4.2, the Clause at line 18, consisting of six literals, will be examined. This clause is in favor of the red player winning, and has been trained by observing completed games:

$$\text{Red}_{\text{win}} = CR_{(3,0),(5,5)} \wedge CR_{(2,2),(1,6)} \wedge CR_{(4,4),(6,6)} \wedge \neg CB_{(4,0),(1,6)} \wedge \neg CB_{(0,1),(5,1)} \wedge \neg CB_{(0,5),(6,6)}$$

Each literal will be analyzed to explain its meaning within the hex board, each **literal** plays a distinct role:

- $CR_{(3,0),(5,5)}$: Indicates a red connection from cell (3, 0) to (5, 5).
- $CR_{(4,4),(6,6)}$: Indicates a red connection from cell (4, 4) to (6, 6).
- $CR_{(2,2),(1,6)}$: Indicates a red connection from cell (2, 2) to (1, 6).
- Negated literals ($\neg CB$): Represent absence of blue connections, such as:
  - $\neg CB_{(4,0),(1,6)}$: Blue has no connection from the cell (4, 0) to (1, 6).
  - $\neg CB_{(0,1),(5,1)}$: Blue has no connection from the cell (0, 1) to (5, 1).
  - $\neg CB_{(0,5),(6,6)}$: Blue has no connection from the cell (0, 5) to (6, 6).

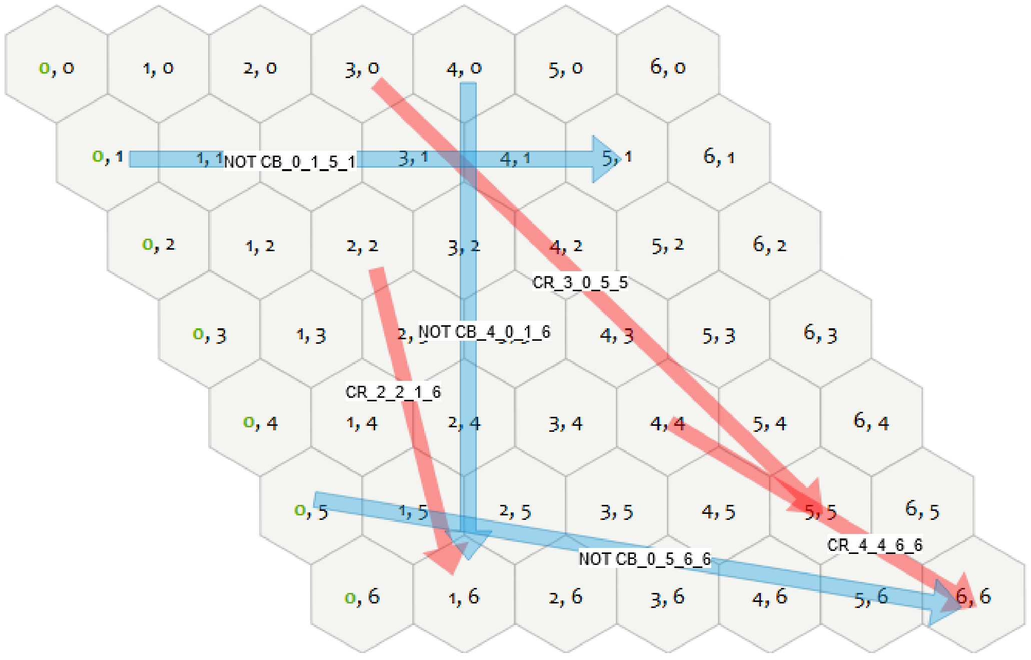


Figure 4.3: Visualization of a red winning clause.

In Figure 4.3, the literal analysis is visualized. For the clause to be evaluated to be true, the following conditions must be satisfied:

- The red player forms a connection starting at cell (3, 0) and ending at (6, 6).
- The literals $CR_{(3,0),(5,5)}$ and $CR_{(4,4),(6,6)}$ combined forms the connection from cell (3, 0) to (6, 6) represented as a red line.
- Negated literals, like $\neg CB_{(0,5),(6,6)}$, show that there is no connection between the two cells here.

The literals $CR_{(3,0),(5,5)}$ and $CR_{(4,4),(6,6)}$ combined makes the blue negated literals unnecessary. This is because $\neg CB_{(4,0),(1,6)}$, $\neg CB_{(0,1),(5,1)}$ and $\neg CB_{(0,5),(6,6)}$ will always evaluate to true when the red player has divided the board by forming a connection from (3,0) to (6,6). These blue connections stated in the literals will never be possible, and one might therefore argue that they are redundant to the clause.

For example, $CR_{(3,0),(5,5)}$ and $CR_{(4,4),(6,6)}$, what's interesting from the two literals is the endpoint cell (5, 5) and starting point cell (4, 4), a question arises: Why did the TM provide two literals for this? and not just one literal displaying a connection from cell (3, 0) to (6, 6)? Maybe this is because it uses two literals to allow for multiple possible routes.

If the literal $CR_{(4,4),(6,6)}$ is satisfied, then $\neg CB_{(0,5),(6,6)}$ will always evaluate to true. This is because red has occupied (6,6), which means that the blue player can never construct this connection.

These interactions demonstrate how the TM identifies and prioritizes winning moves, and the model's weakness of including the negated literals that are redundant.

The literal $CR_{(2,2),(1,6)}$ makes it more difficult for the clause to evaluate to be true. If $CR_{(3,0),(5,5)}$ and $CR_{(4,4),(6,6)}$ evaluate to be true, then the red player has won, but the literal $CR_{(2,2),(1,6)}$ may prevent it from evaluating to be true.

The TM has with the encoded clauses and a literal length of six, provided a deeper insight in how it learns and represents winning patterns in the Hex game by:

1. Identifying critical red connections, such as $CR_{(3,0),(5,5)}$ and $CR_{(4,4),(6,6)}$.
2. Recognizing and representing absent blue connections through negated literals ($\neg CB$).

This analysis illustrates the TM's ability to interpret board configurations and provides an insight into its decision-making process, and choice of symbols to represent the board game using negation to differentiate between blue and red player's cell connections. However the negated blue connections, is considered to be redundant, and the literal $CR_{(2,2),(1,6)}$ makes it less likely that the clause will be evaluated to be true, even though the red player has won, by making a connection from cell (3, 0) to (6, 6). Indicating that there is room for improvement, by decreasing the number of literals that are considered redundant as previously discussed.

4.1.2 Example 2

A clause that has been learned by training on the two moves before the end dataset is:

$\text{Blue}_{\text{win}} = B_{5,5} \wedge CB_{(1,0),(6,0)} \wedge CR_{(2,0),(6,5)} \wedge CR_{(1,1),(5,4)} \wedge CB_{(1,4),(6,8)} \wedge CB_{(4,4),(5,5)} \wedge CR_{(1,5),(3,7)} \wedge \neg CR_{(4,0),(1,4)} \wedge \neg CB_{(3,1),(7,3)} \wedge \neg CB_{(1,2),(7,8)} \wedge \neg CR_{(3,2),(8,3)} \wedge \neg CR_{(4,4),(6,8)} \wedge \neg CB_{(7,4),(0,6)} \wedge \neg CB_{(0,6),(7,7)}$

A visualization for this clause can be viewed in figure 4.4. This clause has one literal, $B_{5,5}$, that states that the blue player has occupied cell $(5,5)$. The other literals in the clause contain information about the connections.

For this clause to evaluate to true, the literal $CB_{(1,0),(6,0)}$ in combination with $CR_{(2,0),(6,5)}$ states that the blue player must have made a relatively long connection. The literal $CR_{(2,0),(6,5)}$ prevents the blue player from creating a short connection directly between $(1,0)$ and $(6,0)$. This is because the red player has divided the board, in some way, between $(2,0)$ and $(6,5)$.

Attempting to make all the literals in the clause to evaluate to become true can be challenging. A board configuration where all the start and end cells in the connection literals have been colored can be viewed in Figure 4.5. The connection literals have to be satisfied without violating the negated connection literals for the clause to be evaluated as true.

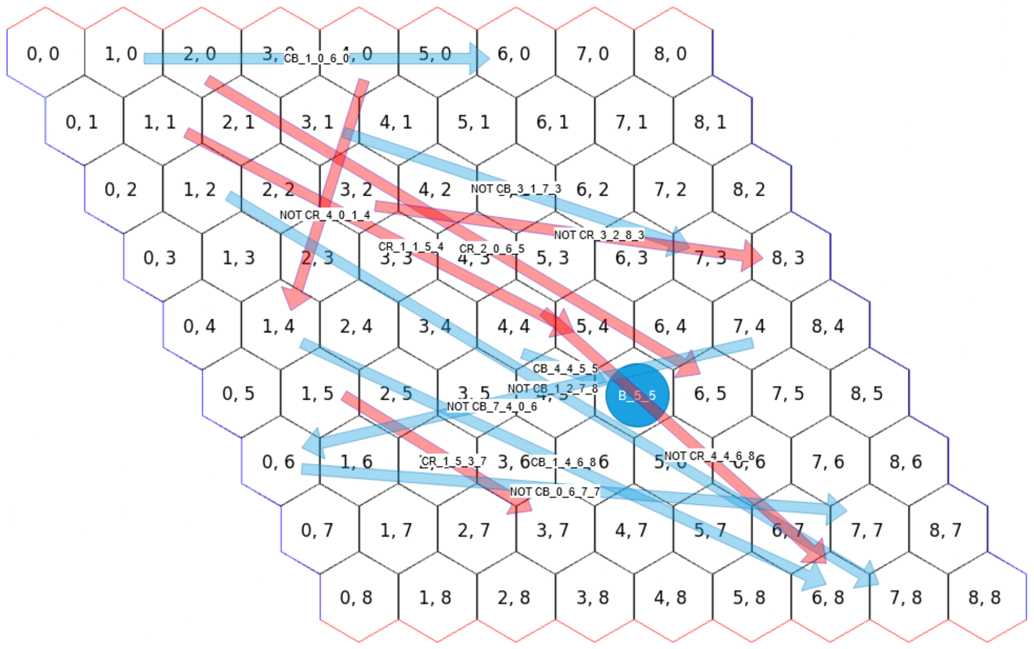


Figure 4.4: Visualization of one of the clauses generated by training on the two moves before the end dataset.

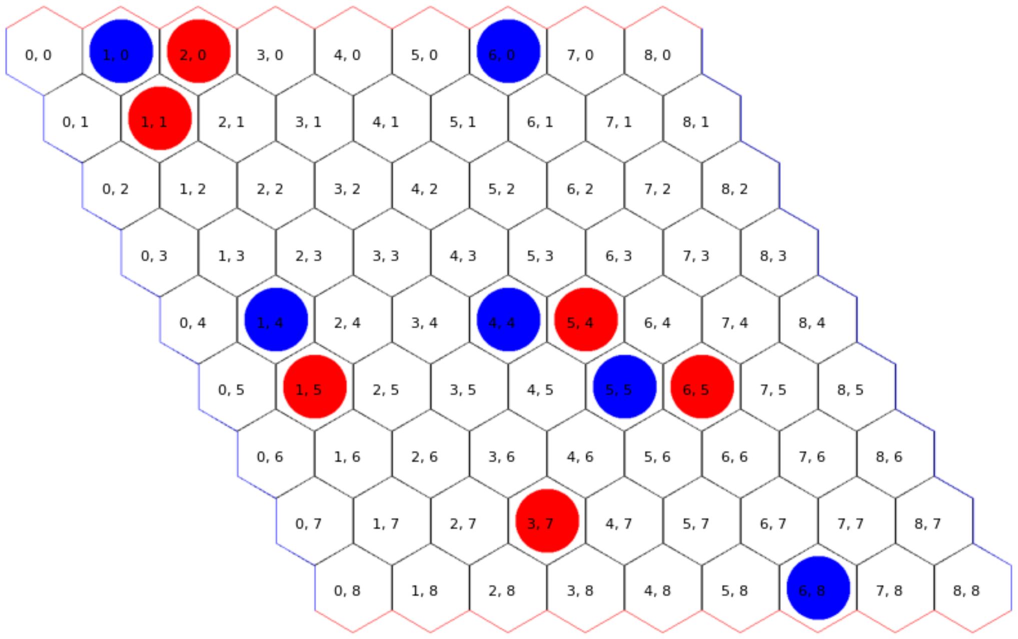


Figure 4.5: Board configuration where the start and end cells from the connection literals have been occupied by the respective players. Puzzle: Color unoccupied cells such that all literals in the clause evaluate to true.

Many of the learned clauses are very simple, and evaluated by themselves, don't necessarily mean that the blue player has won. A few examples of such clauses follow:

$\text{Blue}_{\text{win}} = B_{5,5} \wedge CR_{(1,1),(5,4)} \wedge CR_{(6,5),(4,6)} \wedge \neg CR_{(0,2),(8,4)} \wedge \neg CB_{(4,4),(1,6)} \wedge \neg CB_{(5,6),(3,8)} \wedge \neg CR_{(5,6),(7,8)}$

$\text{Blue}_{\text{win}} = \neg CR_{(2,3),(4,5)}$

$\text{Blue}_{\text{win}} = \neg CB_{(5,1),(7,4)} \wedge \neg CB_{(0,2),(2,7)} \wedge \neg CR_{(0,3),(4,8)}$

Some blue-winning clauses even seem in favor of the red player winning. The exact reason for why this occurs has not been discovered. It may be because the TM is vote-based, meaning that the clauses work together and when all votes are added up it results in the correct prediction. Another reason for why this happens may be found by a closer examination of the training process and the configuration of the hyperparameters, which is a topic for further and future research.

5 Conclusion

A TM-based solution for predicting the winner has been developed. The solution has been tested on three different datasets and discovered to perform well. It achieved greater than 95% accuracy on all datasets that were utilized.

Datasets for two and five moves before the end were acquired by watching and recording two AIs playing against each other. A Python script was developed for this purpose. The collected datasets are quite small, with about 5000 games each. The size likely affects the performance in some way. Some concerns with this approach of acquiring these datasets have been raised. An extension to this project may be to look further into creating and evaluating such datasets.

The game of hex, like chess, is a game of patterns. A thorough analysis of patterns has not been carried out, but the proposed solution has been developed with the existence of patterns such as the bridge in mind. The intention is that it should be able to detect and learn such patterns. Further studies of patterns and incorporation into the solution may yield even better results. A closer look at the studies of solving hex carried out by Ryan Hayward and Philip Henderson [15], [16] may prove beneficial.

One possible issue with the proposed solution is that it does not scale well to bigger boards. A total of $N^2(N^2 + 1)$ symbols are needed. This amounts to a total of 83,810 symbols for boards of size 17x17. One way of decreasing the number of symbols has been proposed, but not tested. Finding ways to decrease the number of symbols is of great interest.

A light analysis of some of the generated clauses has been conducted. The TM can find some interesting ways of combining occupied cells and connections into winning clauses. It is found that many of the learned clauses are too simple to stand alone giving a clear winner. Some reasons for why this occurs have been proposed, but more research is needed to give a conclusive answer.

The generated clauses can be interpreted as a constraint satisfaction problem. Where each of the literals is a constraint that has to be satisfied for the clause to be evaluated as true. The challenge involves coloring the board such that all literals evaluate to true. One clause may have thousands of solutions, and some may have a few solutions, this can be compared to games such as Cross Word, and Sudoku.

All in all the project has been beneficial as an introduction to the TM paradigm and the game of hex. By further investigation into decreasing the number of symbols and utilizing what is known about patterns the proposed solution may scale and perform even better. A TM-based agent may prove to be a fierce competitor against state-of-the-art AI hex game solvers.

References

[1] R. B. Hayward and B. Toft, *Hex: The full story.* CRC Press, 2019.

[2] P. Henderson, "Playing and solving the game of hex," 2010.

[3] C. Hollingsworth, *Game of Hex*, `https://www.kaggle.com/datasets/cholling/game-of-hex`, Dataset available on Kaggle, Nov. 2024. [Online]. Available: `https://www.kaggle.com/datasets/cholling/game-of-hex`.

[4] O.-C. Granmo, "The tsetlin machine–a game theoretic bandit driven approach to optimal pattern recognition with propositional logic," *arXiv preprint arXiv:1804.01508*, 2018.

[5] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O.-C. Granmo, "Learning automata based energy-efficient ai hardware design for iot applications," *Philosophical transactions of the royal society a*, vol. 378, no. 2182, p. 20 190 593, 2020.

[6] R. Saha, O.-C. Granmo, and M. Goodwin, "Using tsetlin machine to discover interpretable rules in natural language processing applications," *Expert Systems*, vol. 40, no. 4, e12873, 2023.

[7] K. Darshana Abeyrathna, O.-C. Granmo, X. Zhang, L. Jiao, and M. Goodwin, "The regression tsetlin machine: A novel approach to interpretable nonlinear regression," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20 190 165, 2020.

[8] O.-C. Granmo, S. Glimsdal, L. Jiao, M. Goodwin, C. W. Omlin, and G. T. Berge, "The convolutional tsetlin machine," *arXiv preprint arXiv:1905.09688*, 2019.

[9] S. J. Otterlei and J. A. Reiersøl, "Playing endgame chess with the tsetlin machine," M.S. thesis, University of Agder, 2020.

[10] A. L. Simonsen and O. A. Haddeland, "Playing the game of hex with thetsetlin machine and tree search," M.S. thesis, University of Agder, 2020.

[11] C. Giri, O.-C. Granmo, H. Van Hoof, and C. D. Blakely, "Logic-based ai for interpretable board game winner prediction with tsetlin machine," in *2022 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022, pp. 1–9.

[12] L. Tautenhahn, *Hex*, [Online; accessed 23. Nov. 2024], Sep. 2024. [Online]. Available: `https://lutanho.net/play/hex.html`.

[13] A. Patel, *Hexagonal grids*, Updated in Mar 2015, Apr 2018, Feb 2019, May 2020, Oct 2021, Dec 2022, Feb 2023, Oct 2024, [Online; accessed 2024-12-13], 2013. [Online]. Available: `https://www.redblobgames.com/grids/hexagons/`.

[14] T. Maarup, "Everything you always wanted to know about hex but were afraid to ask," *University of Southern Denmark*, 2005.

[15] P. Henderson, B. Arneson, and R. B. Hayward, “Solving 8x8 hex,” in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[16] R. Hayward, Y. Björnsson, M. Johanson, M. Kan, N. Po, and J. van Rijswijck, “Solving 7×7 hex: Virtual connections and game-state reduction,” *Advances in Computer Games: Many Games, Many Challenges*, pp. 261–278, 2004.

[17] *Bridge - HexWiki*, [Online; accessed 23. Nov. 2024], Nov. 2024. [Online]. Available: `https://www.hexwiki.net/index.php/Bridge`.

[18] *Interior template - HexWiki*, [Online; accessed 23. Nov. 2024], Nov. 2024. [Online]. Available: `https://www.hexwiki.net/index.php/Interior_template`.

[19] *NetworkX — NetworkX documentation*, [Online; accessed 26. Nov. 2024], Oct. 2024. [Online]. Available: `https://networkx.org`.

[20] *GraphTsetlinMachine*, [Online; accessed 26. Nov. 2024], Nov. 2024. [Online]. Available: `https://github.com/cair/GraphTsetlinMachine`.