Generic heuristic for the `mnk-games`

Abdel-Hafiz ABDOULAYE*, Vinasetan Ratheil HOUNDJI*, Eugène C. EZIN*, Gael AGLIN**

* Institut de Formation et de Recherche en Informatique (IFRI)
Université d'Abomey-Calavi (UAC)
Abomey-Calavi
Bénin
** Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM)
Université catholique de Louvain (UCL)
Louvain la Neuve
Belgique

RÉSUMÉ. Les jeux en situation d'adversité sont très étudiés en intelligence artificielle. Parmi ces jeux, nous distinguons les jeux `mnk`. Un jeu `mnk` est un jeu dans lequel deux joueurs placent, chacun à son tour, une pièce de leurs couleurs respectives sur un plateau $m \times n$. Le vainqueur est le joueur qui obtient en premier un alignement de k pièces de sa couleur sur une ligne soit horizontalement, verticalement ou diagonalement. Pour la résolution de ce type de jeu, des algorithmes de recherche basés sur le parcours d'arbre comme alpha-beta sont utilisés avec des heuristiques spécifiques. Nous proposons une heuristique générique permettant d'évaluer les noeuds des arbres pour les jeux `mnk`. Nous utilisons une méthode d'apprentissage automatique (en particulier l'algorithme Q-learning) pour adapter les différents paramètres. Les résultats des tests montrent qu'en moyenne notre approche est meilleure que certaines heuristiques connues.

ABSTRACT. Adversarial games are very studied in artificial intelligence. Among these games, there are the `mnk-games`. An `mnk-game` is a board game in which two players take turns in placing a piece of their color on an $m \times n$ board. The winner is the player who first gets k pieces of his own color in a row; horizontally, vertically, or diagonally. For the resolution of this type of games, search algorithms based on tree search like alpha-beta are coupled with specific heuristics. In this paper, we propose a generic heuristic to evaluate the moves for `mnk-games`. Then we use a machine learning algorithm (in particular the Q-learning algorithm) to fit the different parameters of the heuristic to each `mnk-game`. This allows us to determine better parameters for the heuristic. For the tests, we associate it with alpha-beta. The experimental results show that our approach is better than some known heuristics.

MOTS-CLÉS : jeux mnk, heuristique générique, apprentissage automatique, Q-learning, alpha-beta.

KEYWORDS : mnk-games, generic heuristic, machine learning, Q-learning, alpha-beta.

1. Introduction

Artificial intelligence is a branch of computer science that aims to understand and build intelligent entities. It is involved in a variety of areas including games. A game is a good testing field for artificial intelligence. For determistic, turn-taking and zero-sum games, some methods based on the tree search as alpha-beta [6, 3] have been proposed to play them more easily and faster. Most of these methods use an evaluation function to improve the final result. Actually the search space of games can be very large and then it is difficult to explore the whole tree in a reasonable time. In this case, the usage of a heuristic represents an alternative. In adversarial search, a heuristic is a function applied on nodes, that evaluates the state of the game by estimating the players gain in order to choose the most promising move.

In this paper we focus on a particular game category, the `mnk-games`. An `mnk-game` [7, 12] is a board game in which two players take turns in placing a piece of their color on an $m \times n$ board. The winner is the player who first gets k pieces of his own color in a row ; horizontally, vertically, or diagonally. We propose a generic heuristic based essentially on the notion of threat (see Section 2). This heuristic makes an evaluation of hits given the parameters associated with the threats. The parameters have static values favoring the choice of the best move at a moment of the game. Firstly, the parameters of the proposed generic heuristic have been set experimentally but it does not always guarantee that they are good for each game. Therefore we use a machine learning algorithm to improve the quality of the parameters of the generic heuristic. We use and experiment an approach which, due to machine learning and specially reinforcement learning, permit to determine the parameters of the generic heuristic mentioned above in order to have better parameters.

This paper is organized as follows : Section 2 gives some theoretical notions used in the paper ; Section 3 explains our generic heuristic and the machine learning method which we use to improve the quality of the different parameters of the heuristic ; Section 4 presents some experimental results ; and Section 5 concludes and provides some perspectives.

2. Background

In this section we define the notion of threat, present some heuristics that use threats, and briefly explain the reinforcement learning.

2.1. Threat

In `mnk-games`, the threat is a very important notion. It represents a configuration of aligned pieces in a certain way that can assure to its player a certain winning trend. It may be advantageous or not because threats of player are always compensated with the threats of second player for the evaluation of the position. Several works use the notion of threat (see for example [1, 4, 14]).

To make this concept more understandable, we will take a `mnk-game` whose purpose is to align 5 pieces. We give the different possible threats in an environment in which the combination of five (05) aligned pieces is winning : the *four*, the *three*, the *two* and the *ones*. A *four* is an alignment of four pieces of one player either horizontally, vertically or diagonally. There are several types of *four* categorized into three categories. Below we present six configurations that give the *four* :

– Type 1 : four consecutive aligned pieces whose extremities are free ;

– Type 2 : four consecutively aligned pieces whose location at one extremity is free while the second is occupied ;

– Type 3 : four consecutive aligned pieces whose extremities are occupied ;

– Type 4 : four pieces aligned with a jump location and whose extremities locations are free ;

– Type 5 : four pieces aligned with a jump location and whose location at one extremity is free while the second is occupied ;

– Type 6 : four pieces aligned with a location jump and whose extremities locations are occupied.

These are the three categories : the *four open* (type 1), the *four half-open* (type 2, 4, 5, 6) and the *four closed* (type 3).

2.2. Heuristic of Shevchenko

Shevchenko [8] does an analysis of the combinations of pieces present on the game board on the lines as well as on the columns and the diagonals. The analysis concerns only the player's pieces. In this sense, Shevchenko only takes into account the threats of one player on the board. Moreover, given k, the number of pieces to align before winning, interresting threats are those of size of k, $k-1$, and $k-2$ with no distinction between half-open, open and closed types. Parameters are associated with the threats according to their size and remain unchanged until the end of the game. These settings are : 100 for the k size threat, 10 for the $k-1$ threat, and 1 for the $k-2$ threat. Shevchenko used his heuristic for Gomoku game.

2.3. Heuristic of Chua Hock Chuan

The application of the Chua Hock Chuan heuristic [5] requires to find the alignments of player's and opponent's pieces on the lines, the columns and the diagonals. It therefore considers the threats of the player and the opponent present on the board but only those of size k, $k-1$ and $k-2$. There is no distinction between half-open and open types. The parameters associated with the player's and opponent's threats are fixed until the end of the game. We have : threat size k (100 for the player and -100 for the opponent) ; threat size $k-1$: 10 for the player and -10 for the opponent ; threat size $k-2$: 1 for the player and -1 for the opponent. Chua Hock Chuan proposed this heuristic for Tic Tac Toe game.

2.4. Reinforcement learning

The reinforcement learning problem is a kind of direct framework of the problem of interaction learning to achieve a goal. The learner or decision maker is called the agent that interacts with its environment. The agent selects the actions and the environment responds and presents new situations to the agent. The environment gives rise to rewards, special numerical values that the agent tries to maximize. A complete specification of an environment defines a task, an instance of the reinforcement learning problem.

Formally, the basis of the reinforcement learning model is : a set of states S of the agent in the environment ; a set of actions A that the agent can perform ; and a set of reward scalar values R that the agent can obtain.

At each step t of the algorithm, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and receives from the environment a new state s_{t+1} and a reward r_{t+1}. Based on these interactions, the reinforcement learning algorithm must allow the agent to develop a $\Pi : S \rightarrow A$ policy that allows him to maximize the amount of rewards. Thus the reinforcement learning method is particularly suited to problems that require a compromise between the quest for short-term rewards and long-term rewards.

If we had to identify a central and new idea to reinforce learning, it would certainly be learning by Temporal Difference (TD) [10, 11, 2]. TD learning is a machine learning method based on

prediction. TD methods use experience to solve the prediction problem. Given some experience following a Π policy, they update their v estimate of v_Π (value obtained by following the Π policy) for non-terminal states s_t occurring in this experiment. A policy is a rule that the agent follows for the choice of actions, given the state in which he is. At the moment $t+1$, they immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The equation (1) gives the update formula.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{1}$$

One of the most important advances in reinforcement learning has been the development of an off-policy TD control algorithm called Q-learning [11, 9, 13]. Q-learning is used to find an optimal action selection policy. It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. When such an action value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. The algorithm has an update formula which calculates the quantity of a state-action combination :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2}$$

Before learning begins, Q returns a fixed (arbitrary) value chosen. Whenever the agent selects an action, observes a reward and a new state (that may depend on both the previous state and the selected action) then Q is updated.

3. Generic heuristic and determination of the different parameters

This section specifies the basic elements for the generic heuristic's definition, gives the formula and describes the method that we propose for parameters determination using reinforcement learning.

3.1. Generic heuristic

For a game in which the number of pieces to line up to win is k, the big threat that needs to be created and that always leads to a win is the $k-1$ open type and the only combination that can come up in this configuration is the $k-2$ open type threat. On the other hand, the half-open threat of $k-2$ is not interesting in itself. Also, the $k-1$ half-open threat is very close to victory, it is less important than a $k-2$ open-car type. In the latter case, we can open $k-1$ half-open type as it can also lead to the $k-1$ open threat , which is very interesting. In general, the game becomes decisive when on threats of size $k-2$ and $k-1$. Threats smaller than $k-2$ are not interesting. These are threats that are not co-affected by the same weighting as those that are adverse or not. A large weighting is a given to the threats against the player's threats to prevent the opponent from taking an irreversible advantage. On the other hand, threats classified as uninvolved (less than $k-2$ and the half-open threat of $k-2$) are co-assigned in the same way as players.

Below we present the formula of the proposed generic heuristic based on the threats defined above. The different values of the parameters of the heuristic were first fixed regarding the priority of threats and experimentations.

$$A = \begin{cases} \sum_{i=1}^{k-3} (a_{2i-1}p_{i,1} + a_{2i}p_{i,2}) + a_{2(k-2)-1}p_{k-2,1} + 100p_{k-2,2} + 80p_{k-1,1} + 250p_{k-1,2} + 1000000p_k & \text{if } k > 3 \\ a_1 p_{k-2,1} + 100p_{k-2,2} + 80p_{k-1,1} + 250p_{k-1,2} + 1000000p_k & \text{if } k = 3 \end{cases}$$

$$B = \begin{cases} \sum_{i=1}^{k-3} (a_{2i-1}q_{i,1} + a_{2i}q_{i,2}) + a_{2(k-2)-1}q_{k-2,1} + 1300q_{k-2,2} + 2000q_{k-1,1} + 5020q_{k-1,2} + 1000000q_k \text{ if } k > 3 \\ a_1 q_{k-2,1} + 1300q_{k-2,2} + 2000q_{k-1,1} + 5020q_{k-1,2} + 1000000q_k \text{ if } k = 3 \end{cases}$$

$$f = A - B$$

in which A is the evaluation of the player's threats on the board ; B is the evaluation of the opponent's threats on the board ; a_i is the coefficient of the lower threat index i ; $p_{i,1}$ is the player's number of half-open threats of size i ; $p_{i,2}$ is the player's number of open threats of size i ; p_i is the player's number of threats without hole of size i ; $q_{i,1}$ is the opponent's number of half-open threats of size i ; $q_{i,2}$ is the opponent's number of open threats of size i : q_i is the pponent's number of threats without hole of size i : n is the number of alignment leading to victory.

3.2. Proposed method for automatic determination of parameters

The generic heuristic is based on the notion of threat. The parameters of the heuristic are assigned according to the importance of the threats and are the same for each game. The best option is to be able to determine the parameters adapted to the situations encountered during the resolution of the game and to the rules of displacement. In this setion we propose a way to automatically update the parameters of the heuristic with the Q-learning method.

Q-learning uses a quality evaluation function Q. This allows to have a table of values (Q values) that helps in the choice of an action when we are in a given state. The value of the parameters greatly influences the evaluation of a position and the choice of the move to play. Therefore, we consider that using Q-learning, we must evaluate the quality of the parameters of the heuristic. The Q values are the parameters to be determined. Before the learning begins, the Q function returns a fixed value chosen by the programmer. We recall that the heuristic used fixed parameters determined experimentally and which proved their worth. These are the parameters that we use as initial Q values.

The algorithm allows to update a value by time step, a parameter in our case. It is necessary to find the parameter to update for a given step. We get a new step when a move is made. The alpha-beta algorithm returns the best estimated move to go to the next step. We look for all the threats on the board after a move and identify the most important threat. The parameter associated with the latter is the update. Also we associate a reward to each type of threat since the chosen action leads to a certain configuration of threats. In general, it is null except for the goal state (state of the board where the player has the "n" required pawns aligned). Identifying the most important threat allows you, at this stage, to know which reward to use for the update.

The actual update is done using the formula (2) and requires finding the maximum value Q in the next state. It will be necessary to determine the types of threat that the legal movements could create in order to take the maximum of their parameters. If we have an S state that, after an A action, leads to an S' state, we simulate the possible moves from the S' state, collect the most important threats likely to be created after each move and identify the most important of them. Its parameter is the maximum value Q sought.

The evaluation of a position takes into account the threats of the player and the opponent. As a result, we also update the threat parameters of the opponent to avoid any bad evaluation of the heuristic. This update is done following the same principles that we described for the player. The parameters are the initial Q values and we identify the largest enemy threat created by an action. The Q function is used for the corresponding parameter and requires to find the maximum value of the parameters for the most important enemy threats obtained after the simulation from the S' state.

We make two updates at each step : one on the parameter associated with the most important threat of the player and the other on the parameter associated with the most important threat of the opponent.

In addition, the learning rate and the reduction factor of the update formula have a large impact on the learning process. Since the game play environment is entirely deterministic, we chose a learning rate of 1. The reduction factor was chosen to meet the basic requirements of heuristics. A value too close to 1 would make the threat coefficients too close to each other. This leads to a bad evaluation of the positions and thus makes the heuristic less efficient. A value close to 0 ensures to a certain extent a favorable difference between the parameters. After many tests, we selected $\gamma = 0.1$. The update formula (2) becomes :

$$Q(S, A) = R + \gamma maxQ(S', a) \tag{3}$$

We use Algorithm 1 to update the parameters of the heuristic.

Algorithm 1: Parameters Update Algorithm

INITIALIZE THE TABLE OF VALUES WITH THE EXISTING FIXED PARAMETERS
repeat
  Initialize S with the current state of the board
  repeat for each step of the episode
    Choose A `// action from alpha-beta algorithm using generic heuristics`
    Execute the action A
    Find the parameter P_P associated with the player's most important threat
    Find the parameter P_O associated with the opponent's most important threat
    Receive the reward R_P corresponding to the player's most important threat
    Receive the reward R_O corresponding to the opponent's most important threat
    Observe the following state S'
    $P_P(S, A) \leftarrow R_P + \gamma maxP_P(S', a)$ `//` $maxP_P(S', a)$ `is the maximum value of parameters for the most important player's threats that can be created from the new state` S'
    $P_O(S, A) \leftarrow R_O + \gamma maxP_O(S', a)$ `//` $maxP_O(S', a)$ `is the maximum value of parameters for the most important opponent's threats that can be created from the new state` S'
    $S \leftarrow S'$
  until *S terminal*
until *the end of the episode*

At each stage of the episode, the alpha-beta algorithm uses the heuristic with the new parameters for actions selection.

4. Experimental results

For the experiments we consider an agent who uses the alpha-beta algorithm and the generic heuristic (heuristic with the parameters automatically updated with the Q-learning) to a depth limit of 4. We conducted numerous tests to determine the time that would allow the proposed solution to provide a set of stable parameters. After analyzing the results, we make the intelligent agent play against itself for 5 episodes because from this level, the parameters seem not to change anymore and each of them seems to have been updated at least once.

For Gomoku, we check the performance by playing against a player who uses alpha-beta and Shevchenko's heuristic [8]. For Tic tac toe, we compare the improved heuristic and Chua Hock Chuan's heuristic [5].

We conducted several tests grouped into categories. For each game considered, we distinguish three (03) categories based on the number of pieces to be aligned to win the game : category 1 with $k = 3$; category 2 with $k = 4$; and category 3 with $k = 5$. For each category, we vary the size of the board with $m = n$ and the tests are done for according to the player who starts the game. The maximum size is 11.

4.1. Gomoku : Improved heuristic vs Shevchenko's heuristic

4.1.1. Category 1 : $k = 3$

See Table 1 and Table 2.

With $k = 3$ the first player always wins the game no matter the size of the board with one exception because for a size of 7 our approach wins the party as a second player. It defended herself and created advantageous situations. We conclude that for $k = 3$, the first player has an advantage that leads him to victory. For better analysis, we change k.

4.1.2. Category 2 : $k = 4$

See Table 3 and Table 4.

With $k = 4$, our approach always wins the game it plays first or not for a size bigger than k. It is very effective at this level.

4.1.3. Category 3 : $k = 5$

See Table 5 and Table 6.

With $k = 5$, our heuristic always wins the game when it plays first or not for a size over k. It is more efficient than Shevchenko's heuristic even if the game ends with a draw when it plays second for $k = 5$.

According to those three analysis, we conclude that the player using the improved heuristic won the most games regardless of the number of pieces to be aligned and the size of the board. Our heuristic has proven its efficiency against Shevchenko's heuristic.

4.2. Tic tac toe : Generic heuristic vs Chua Hock Chuan's heuristic

4.2.1. Category 1 : $k = 3$

See Table 7 and Table 8.

As second player, the generic heuristic is totally out of date and is not effective as first player.

4.2.2. Category 2 : $k = 4$

See Table 9 and Table 10.

Analysis 5 : With $k = 4$, our approach always wins the game no matter it plays first or not for a size bigger or equal to 6. It concedes no defeat. This number of pieces to align is favorable.

4.2.3. Category 3 : $k = 5$

See Table 11 and Table 12.

Our approach always wins the game no matter it plays first or not for a size bigger than or equal to 8. It concedes no defeat. Just like $k = 4$, $k = 5$ is good for our approach.

The Chua Hock Chuan's heuristic [5] showed the effectiveness of the latter for $k = 3$ but our approach was successful for $k > 3$.

5. Conclusion and perspectives

In this work, we have determined a generic evaluation function for `mnk-games` to solve all games in this category with the same solution. Then we have proposed a method for determining the parameters of the generic heuristic using machine learning to obtain better parameters for each game considered. It is based on the functioning of Q-learning, which is an "off-policy" TD control algorithm. We combined the improved heuristic, Shevchenko's heuristic and Chua Hock Chaun's heuristic with the alpha-beta algorithm to make a comparison on Gomoku and Tic tac toe. The results of the tests showed that at different levels the improved heuristic is on average more efficient.

As future works we would like to intensively compare our generic heuristic on a large variety of `mnk-games` wrt other heuristics. A good perspective of this work is to determine some elements to improve the approach proposed by focusing on the eligibility traces associated with TD methods [4]. We can also automatically learn the winning strategies used in games played for each `mnk-game`.

6. Bibliographie

[1] L.V. ALLIS, H.J. VAN DEN HERIK, M.P.H. HUNTJENS, Go-moku solved by new search techniques, Computational Intelligence, 12(1), p.7-23., 1996.

[2] ANDREW G. BARTO, Temporal difference learning, *www.scholarpedia.org/article/Temporal_difference_learning*, 2007.

[3] TRISTAN CAZENAVE, Des Optimisations de l'Alpha-Beta, Laboratoire d'Intelligence Artificielle, Département Informatique, Université Paris 8, 2011.

[4] TRISTAN CAZENAVE, A Generalized Threats Search Algorithm, International Conference on Computers and Games. Springer, Berlin, Heidelberg, 2002.

[5] CHUA HOCK CHUAN, Java Games, *http ://www3.ntu.edu.sg/home/ehchua/programming/java /JavaGame_TicTacToe_AI.html*, 2017.

[6] SAMUEL H. FULLER, JOHN G. GASCHNIG, *Analysis of the alpha-beta pruning algorithm*, Department of Computer Science, Carnegie-Mellon University, 1973.

[7] H.J VAN DEN HERIK, J.W.H.M UITERWIJK, J.V RIJSWIJCK, Games solved : Now and in the future, *Artificial Intelligence*, Vol. 134, p. 277-311, 2002.

[8] MYKOLA SHEVCHENKO, GOMOKU & Minimax-alphabeta search, *https ://github.com/nshevchenko/GomokuAlphabeta/blob/master/gomoku-minimax-alphabeta.pdf, 2016.*

[9] OLIVIER SIGAUD, OLIVIER BUFFET, *Markov Decision Processes in Artificial Intelligence*, The MIT Press Cambridge, Massachusetts, 2010.

[10] RICHARD S. SUTTON, *Learning to Predict by the Method of Temporal Differences*, Machine Learning, vol.3, p.9-44, 1988.

[11] RICHARD S. SUTTON, ANDREW G. BARTO, Reinforcement Learning : An Introduction, *The MIT Press Cambridge*, Massachusetts, 2012.

[12] J.W.H.M UITERWIJK, H.J VAN DER HERIK, *The advantage of the initiative*, Information Sciences, p. 43-58, 2000.

[13] CHRISTOPHER WATKINS, PETER DAYAN, *Q-learning*, Machine Learning, p. 279-292, 1992.

[14] I-CHEN WU, DEI-YEN HUANG, A New Family of k-in-a-row Games, Advances in Computer Games, 2005.

7. Appendix

7.1. Gomoku experimental results

7.1.1. Category 1 : $k = 3$

Tableau 1. *Gomoku : Generic heuristic vs Shevchenko's heuristic for $k = 3$, test n°1*

Players	First Player	Taille $\in$ [3,11]
Alpha-beta + generic heuristic	Yes	Win
Alpha-beta + heuristic of Shevchenko	No	Loss

Tableau 2. *Gomoku : Generic heuristic vs Shevchenko's heuristic for $k = 3$, test n°2*

Players	First player	Taille $\in \{3,4,5,6,8,9,10,11\}$	Taille = 7
Alpha-beta + generic heuristic	No	Loss	Win
Alpha-beta + heuristic of Shevchenko	Yes	Win	Loss

7.1.2. Category 2 : $k = 4$

Tableau 3. *Gomoku : Generic heuristic vs Shevchenko's heuristic for $k = 4$, test n°3*

Players	First player	Taille = 4	Taille $\in$ [5,11]
Alpha-beta + generic heuristic	Yes	Draw	Win
Alpha-beta + heuristic of Shevchenko	No	Draw	Loss

Tableau 4. *Gomoku : Generic heuristic vs Shevchenko's heuristic for $k = 4$, test n°4*

Players	First player	Taille = 4	Taille $\in$ [5,11]
Alpha-beta + generic heuristic	No	Draw	Win
Alpha-beta + heuristic of Shevchenko	Yes	Draw	Loss

7.1.3. Category 3 : $k = 5$

Tableau 5. *Gomoku : Generic heuristic vs Shevchenko's heuristic for $k = 5$, test n°5*

Players	First player	Taille $\in$ [5,11]
Alpha-beta + generic heuristic	Yes	Win
Alpha-beta + heuristic of Shevchenko	No	Loss

Tableau 6. *Gomoku : Generic heuristic vs Shevchenko's heuristic for $k = 5$, test n°6*

Players	First player	Taille = 5	Taille $\in$ [6,11]
Alpha-beta + Generic heuristic	No	Draw	Win
Alpha-beta + heuristic of Shevchenko	Yes	Draw	Loss

7.2. Tic tac toe experimental results

7.2.1. Category 1 : $k = 3$

Tableau 7. *Tic tac toe : Generic heuristic vs Chua Hock Chuan's heuristic for $k = 3$, test n°7*

Players	First player	Size = 3	Size = $\{4,6,7\}$	Size = $\{5,8,9,10,11\}$
Alpha-beta + generic heuristic	Yes	Draw	Loss	Win
Alpha-beta + heuristic of Chua Hock Chuan	No	Draw	Win	Loss

Tableau 8. *Tic tac toe : Generic heuristic vs Chua Hock Chuan's heuristic for $k = 3$, test n°8*

Players	First player	Size = 3	Size $\in$ [4,11]
Alpha-beta + generic heuristic	No	Draw	Loss
Alpha-beta + heuristic of Chua Hock Chuan	Yes	Draw	Win

7.2.2. Category 2 : $k = 4$

Tableau 9. *Tic tac toe : Generic heuristic vs Chua Hock Chuan's heuristic for $k = 4$, test n°9*

Players	First player	Size = $\{4,5\}$	Size $\in$ [6,11]
Alpha-beta + generic heuristic	Yes	Draw	Win
Alpha-beta + heuristic of Chua Hock Chuan	No	Draw	Loss

Tableau 10. *Tic tac toe : Generic heuristic vs Chua Hock Chuan's heuristic for $k = 4$, test n°10*

Players	First player	Size = $\{4,5\}$	Size $\in$ [6,11]
Alpha-beta + generic heuristic	No	Draw	Win
Alpha-beta + heuristic of Chua Hock Chuan	Yes	Draw	Loss

7.2.3. Category 3 : $k = 4$

Tableau 11. *Tic tac toe : Generic heuristic vs Chua Hock Chuan's heuristic for $k = 5$, test n˚11*

Players	First player	Taille $\in$ [5,7]	Taille $\in$ [8,11]
Alpha-beta + generic heuristic	Yes	Draw	Win
Alpha-beta + heuristique de Chua Hock Chuan	No	Draw	Loss

Tableau 12. *Tic tac toe : Generic heuristic vs Chua Hock Chuan's heuristic for $k = 5$, test n˚12*

Players	First player	Taille = $\{5,6\}$	Taille $\in$ [7,11]
Alpha-beta + generic heuristic	No	Draw	Win
Alpha-beta + heuristic of Chua Hock Chuan	Yes	Draw	Loss