Science Publications

Journal of Mathematics Letters, 2022, 1, 496
www.scipublications.org/journal/index.php/jml
DOI: 10.31586/jml.2022.496

Article

Notes about Winning Strategies for Some Combinatorial Games

Halyna P. Cherniichuk [1], Ivan H. Krykun [1, 2,*]

[1] Department of Applied Mathematics, Vasyl' Stus Donetsk National University, Vinnytsia, Ukraine

[2] Department of Theory of Control Systems of Institute of Applied Mathematics and Mechanics of NAS of Ukraine, Sloviansk, Ukraine

*Correspondence: Ivan H. Krykun (iwanko@i.ua)

Abstract: We study the theory of combinatorial games and find winning strategies for players. The algorithmic implementation of the winning strategies for the game TacTix is presented and the software implementation for this game in Python programming language is implemented. The program has a console interface and allows one to check the winning strategies in practice.

Keywords: Combinatorial Game Theory, Greedy algorithm, Symmetric algorithm, TacTix, Nim, Winning Strategies.

1. Introduction

Combinatorial game theory is a mathematical theory that studies two-person games where at each moment in time there is a position that the players change accordingly to achieve victory. This theory does not study games involving chance.

Combinatorial games are games played by two players without random moves and with complete information, i.e. all moves are known to both players. Players take turns making moves, thus changing positions from one to another, until the final position is reached, in which there are no possible moves. When the final position is reached, one of the players is declared the winner, and the other - is the loser.

Combinatorial games as a subject of research began to attract the attention of mathematicians in the early twentieth century. The beginning of the study was the article by Charles Bouton [1].

Game theory is applied in various fields of human activity: politics, biology, cybernetics, economics, military affairs, etc. Each player has strategies that he can apply. The interaction of two players creates a situation where each player gets a certain result: win or lose. When choosing a strategy, it is necessary to take into account not only the maximum gain for yourself, but also the possible steps of the opponent and their impact on the situation as a whole [2, 3].

Combinatorial games have important features that distinguish them from other games (i.e. gambling, cooperative, antagonistic, and others). Namely [3]:

- There are two players who take turns;
- There are no random factors - like shuffling cards or using dice;
- There is complete information - both players know all possible moves;
- The game must end with one of the players winning, there can be no draw.
- The last move determines the winner;
- There are two types of the game: "normal game": the player who made the last move wins; "miser game": the player who made the last move loses.

How to cite this paper:
Cherniichuk, H. P., & Krykun, I. H. (2022). Notes about Winning Strategies for Some Combinatorial Games. *Journal of Mathematics Letters*, *1*(1), 1–9. Retrieved from https://www.scipublications.com/journal/index.php/jml/article/view/496

Received: September 06, 2022
Accepted: October 24, 2022
Published: October 26, 2022

2. Model of combinatorial game TacTix

TacTix is a combinatorial game, which is a derivative of the more general Nim game, invented by P. Hayne [4]. It refers to logical or mathematical puzzles. TacTix is a geometric variant of the more general game Nim [3, 5]. The game is played on a board of size N x N, filled with some chips (coins, stones). In one move, a player can pick up several chips from a row or column. The number of chips taken in one move cannot be more than the number of chips in the row or column. Also, the selected chips must be located next to each other, in the same row or column.

There are winning strategies for players (for "normal" game) [5]:

- For the first player: if N is odd, take the centerpiece and copy every move of the opponent symmetrically. Eventually the player takes the last piece and wins.
- For the second player: if N is even, then you need to copy the opponent's moves symmetrically. Eventually, the player will take the last chip and win.

TacTix refers to unbiased games. An unbiased game is an equal combinatorial game with the following properties: the action or move that is available to a player depends only on the state of the game, but does not depend on who makes the move; if a player is unable to make a move, he loses; the game must always end [5]. The termination condition can be changed to obtain the so-called miner's game - the player who made the last move loses. The impartiality condition makes it unimportant who makes the move; each position has a certain outcome taking into account who made the move.

The TacTix game has the property of being finite [5]. A finite game is a game that starts from any position, and the final result is always reached after a certain number of moves, which depends only on the initial position. Thus the game ends when there are no pieces left on the board. Therefore, the number of moves played is equal, at most, to the number of chips on the board.

3. Winning strategies for Nim-like games

TacTix is one of the Nim-like games. The differences are that the game is played on a board of size N x N, in one move a player can take several adjacent chips from a row or column, but not more than the number of chips in a row or column.

The analyzed studies [3, 4, 6] highlight general approaches to Nim-like games. These approaches are based on the rules of the Nim game: there are several rows or piles of chips (in the classic version - 3 piles); any number of items (more than zero) from one pile can be taken in one move. In the "normal" game the player who took the last item wins, in the miser game this player loses.

Different rules for removing chips from the playing field make it possible to divide Nim-like games into several types. We give an overview of existing results for Nim-like games using [6].

Game I. For each move a player can remove from 1 to 3 chips from the board.

Theorem 1. *For game I, the game is winning for the second player at each stage if the number n of chips remaining on the board at this stage is a multiple of 4, and winning for the first player in all other cases. The winning strategy for the first player is to remove n mod 4 chips (the number of chips that is divisible by 4).*

Game II. A player can take any number of chips less than or equal to the number of chips taken by the opponent on the previous turn. The first player can remove any number of chips, except all at once.

The analysis of winning strategies for this type of game shows that this type of game can be reduced to the analysis of binary decomposition of numbers. Charles Bouton in his article [1] described the analysis of the game of Nim using binary decomposition. Each combination of chips was called "dangerous" or "safe". If the position after the next move of the player guarantees him a win, then this position is safe, otherwise the position is unsafe. To determine whether a position is safe or not, the number of chips in each row must be written in binary. If the sum in each column (digit) is zero or even, then the position is safe. If the sum in at least one column is odd, then the position is unsafe.

In this type of Nim-like games, if a player picks up 1 chip at a certain stage, then in the following moves players can pick up only 1 chip. Since according to the rules of the "normal" game the one who takes the last chip wins, the player at the stage when the number of chips remaining on the board becomes odd can win the game by taking 1 chip.

What should the first player do if the number of remaining chips is even? If he takes only 1 chip, he loses the game. If he takes 2 chips, the next moves of the players will be a repetition of taking 2 chips. Therefore, the first player at the stage when the number of remaining chips is an odd number or a multiple of 2 can win the game by taking 2 chips.

If the number of remaining chips is an even number or a multiple of 2, in particular a multiple of 4, then the moves of the players starting from this stage will be a repetition of taking 4 chips, and therefore the player starting the game at a stage when the number of remaining chips is an odd number or a multiple of 4 can win the game by taking 4 chips.

Theorem 2. *In Game II, at the stage when there are* $\mathbf{2^p(2p+1)}$ *chips left on the board, the game is won by the first player if and only if the rules of the game allow to take* $\mathbf{2^p}$ *chips at this stage.*

An example of using Theorem 2: a game according to the rules of Game II, started with $\mathbf{2^p}$ chips on the board, is always won by the second player, since the player who started the game cannot remove all the chips. A Game II game started with a different number of chips than $\mathbf{2^p}$ is won by the player who started the game.

Thus, the winning strategies for Game II were determined. However, it is possible to derive these strategies using another idea that can be applied to games with more complex rules.

Since the number of balls that can be removed from the playing field is constantly changing during the game, we define a number *w(n)*, which denotes the smallest number of balls that must be removed to win at the stage when there are n chips left.

Theorem 3. *The winning strategy for a player who starts Game II with n chips on the board is to take away such a number of chips that the number of chips on the board m, which became after the first player's move, in the binary expansion would have zero in the place of the rightmost one in the binary expansion of the number n.*

Game III. A player can remove from the playing field the number of chips that is less than or equal to twice the number of chips taken by the opponent in the previous turn. The first player can remove any number of chips, except all at once.

For this game, the sequence of Fibonacci numbers plays an important role in finding winning strategies.

Theorem 4. *In game III, at the stage when the number of chips remaining on the board is n, the game is won by the first player if and only if the rules of the game allow to take l(n) chips, where l(n) is some number from the Fibonacci sequence corresponding to the number n (for a detailed explanation see [6]).*

4. Winning strategies for the TacTix game

Given that the TacTix game is a variant of the Nim game, the analysis of the above results suggests that they remain valid for TacTix as well. However, the winning strategies for TacTix are much simpler than for Nim. Therefore, the analysis of different types of the game (given above) was carried out, on the basis of which the winning strategies for TacTix were formulated, and the software implementation of the obtained strategies was performed.

The standard game of TacTix is played on a 4 x 4 playing field filled with chips. Thus, the player can pick up from 1 to 4 adjacent chips in the same row or column.

Let's consider a variation of the rules - Game I from the previous paragraph for the game TacTix. Let's explore how the winning strategies are determined depending on the number of chips remaining on the playing field.

Stages in which the number of chips is $n = 1; 2; 3$ are winning for the first player, since the player can take all the chips remaining on the playing field in one row or column at a time, provided that these chips are located next to each other.

The stage with $n = 4$ remaining chips is winning for the Second player, since any move of the First player leads to a winning situation for the Second player.

At the stage $n = 5$ chips, if the first player takes 2 or 3 chips, then 3 or 2 chips are left and thus the opponent wins.

In addition to the strategies presented in Theorems 1 – 4, the so-called "greedy" or "symmetric" algorithms can also be used for the TacTix game. They are easier to apply, to some extent intuitive, but it is worth remembering the need to adjust the strategy during the game.

The greedy algorithm [7] is a simple and straightforward algorithm that makes the best decision based on the data available at this stage. The player who applies this algorithm does not worry about possible consequences, but hopes to get the optimal solution. In other words, the player tries to take as many chips as possible from the field in one move.

It is advisable to use this algorithm at the first attempt to play a new game, especially if the opponent is an expert. Thus, you will not waste time searching for successful moves. But it is worth noting that this algorithm does not always work. For games such as TacTix and Nim, the use of this algorithm may not lead to an optimal solution, but on the contrary give the worst possible solution.

Symmetric algorithm [7] is an intuitively obvious strategy. Every time the opponent makes a move in one part of the board, the player mirrors this move in the other part of the board.

To successfully use the symmetric algorithm of the game, it is not necessary to leave the opponent a move that will eliminate the possibility of repeating the move (for example, in the center of the board).

5. Software implementation of the game TacTix

Using the above algorithms/strategies, we have created a software implementation of the game TacTix. This implementation is written in Python programming language and has a console interface. The game is played according to the rules of Game I and is played against the computer. All rules are described at the beginning of the game. Also, for convenience, there are hints about possible moves and their number, warnings about the absence of a move, or an erroneous move.

The size of the playing field is 4 x 4. One randomly determines who makes the first move (the player or the computer), next moves will take place in turn.

At the beginning of the game, the player is asked to choose his game symbol ("X" or "O"). After selecting the game symbols, the screen displays an empty field and prompts the player to make a move (if the player moves first) or displays the field with the computer's first move and prompts the player to make his move (otherwise). According

to the selected symbol "X" or "O", the player and the computer take turns filling the whole board.

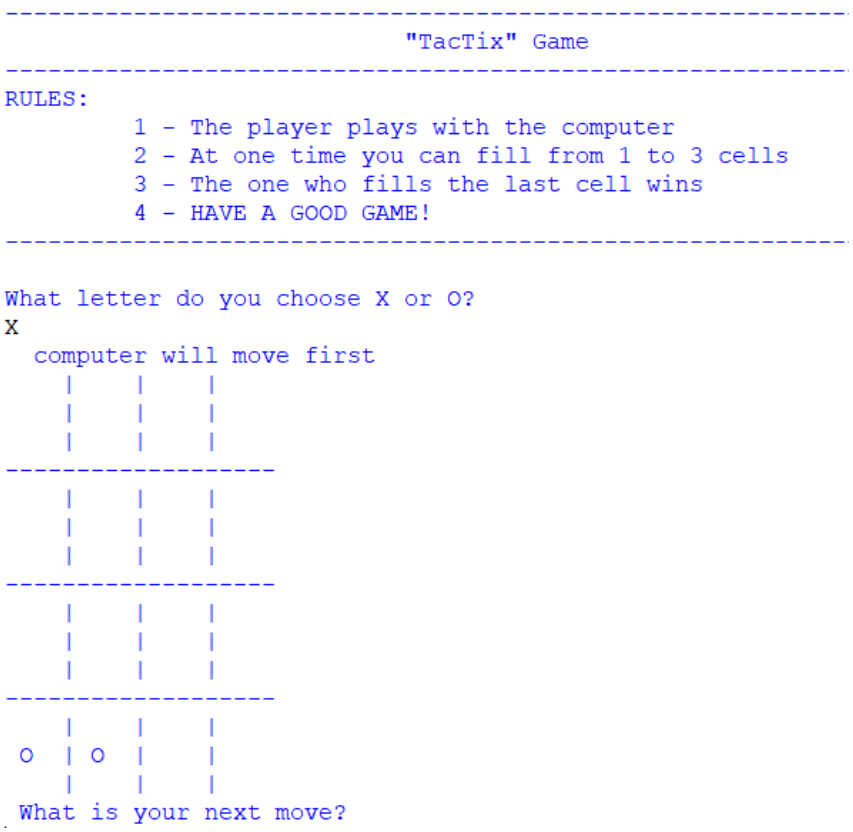

Figure 1. Screenshot of the start of the game with the first computer move

Computer's moves are made according to the described winning strategy (if there is a winning situation for the computer) or to prevent the possibility of implementing the winning strategy for the player (otherwise).

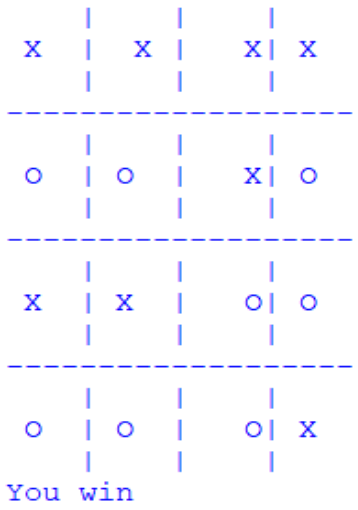

Figure 2. Screenshot of the end of the game

We have developed an algorithm for the game, which is shown in the flowchart below.

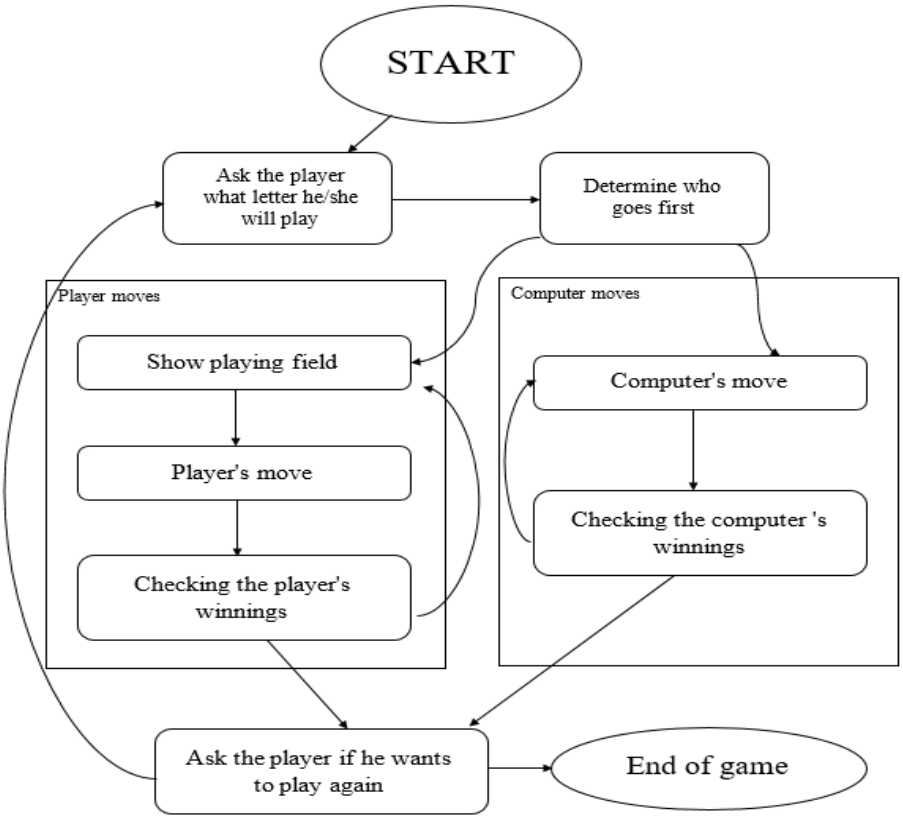

Figure 3. The flowchart for the implementation of the game TacTix

The program code for this implementation of the game TacTix (according to Game I rules) is given in Appendix A.

6. Conclusions and prospects for further research

The developed model of the game TacTix (according to the rules of the Game I) shows in practice the application of the algorithm. During the game, you can clearly see the effectiveness of the strategy, as well as identify situations when the player wins for sure or when one wrong move leads to a computer win.

This model of the game will be further improved, namely the ability to choose the size and shape of the board to make the game more interesting and more complex. Also in the future, it will developed algorithms for the game TacTix according to the rules described as Game II and Game III.

The proposed model of the combinatorial game will be enhanced using recent studies of random processes [8–16] for study games involving chance. About the application of combinatorial games, there is a lot of information in [2, 3, 7, 15, 17, 18].

Author Contributions: All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors express their heartfelt gratitude to the brave soldiers of the Ukrainian Armed Forces who protect the lives of the authors and their families from Russian bloody murderers since 2014.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Program code for the game

```
# "TacTix" Game
import random
print("""
------------------------------------------------------------------------------
                            "TacTix" Game
------------------------------------------------------------------------------
RULES:
         1 - The player plays with the computer
         2 - At one time you can fill from 1 to 3 cells
         3 - The one who fills the last cell wins
         4 - HAVE A GOOD GAME!
------------------------------------------------------------------------------
""")
def drawBoard(board):
    print ('    |    |    |')
    print (' '  + board[13] + '  |  '  + board[14] + ' |   ' + board[15] + '| ' + board[16])
    print ('    |    |    |')
    print ('-------------------')
    print ('    |    |    |')
    print (' '  + board[9]  + '  | ' + board[10]  + '  |   ' + board[11] + '| ' + board[12])
    print ('    |    |    |')
    print ('-------------------')
    print ('    |    |    |')
    print (' '  + board[5]  + '  | ' + board[6]   + '  |   ' + board[7]  + '| ' + board[8])
    print ('    |    |    |')
    print ('-------------------')
    print ('    |    |    |')
    print (' '  + board[1]  + '  | ' + board[2]   + '  |   ' + board[3]  + '| ' + board[4])
    print ('    |    |    |')
def inputPlayerLetter():
    # The player chooses which label (letter) will fill the cells
    letter = ''
    while not(letter == 'X' or letter == 'O'):
        print ('What letter do you choose X or O?')
        letter = input().upper()
    # The first element corresponds to the player and the second element corresponds to the
computer
        if letter == 'X':
            return ['X','O']
        else:
            return ['O','X']
def whoGoesFirst():
    if random.randint(0,1) == 0:
        return 'computer'
    else:
        return 'player'
def playAgain():
    # Function for replay (when the player wants to play again)
    print(' Would you like to play again? (yes or no)')
    return input().lower().startswith('y')
def makeMove(board, letter, move):
    board[move] = letter
def isWinner(board, letter, getComputerMove):
    # The loser is the one who fills in the last box
    if isBoardFull(board):
        return True
    return False
def getBoardCopy(board):
    dupeBoard = []
    for i in board:
        dupeBoard.append(i)
    return dupeBoard
def isSpaceFree(board,  move):
    return board[move] == ' '
def getPlayerMove(board, move_index, previous_moves):
    def isValidMove(move):
        moves = previous_moves + [move]
        return all( (m-1) // 3 == (moves[0]-1) // 3 for m in moves ) or all( m % 3 == moves[0]
% 3 for m in moves)
    move = ' '
    possible_moves ='1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16' .split()
    while  move  not  in  possible_moves  or  not  isSpaceFree(board,  int(move))  or  not
isValidMove(int(move)):
        if move_index == 0:
            print (' What is your next move?')
        else:
            if move in possible_moves and not isValidMove(int(move)):
                print('You can select only those cells that are in the same row or column as
already selected cells ')
            text_cells = 'cells'
            if 3 - move_index == 1:
                text_cells = ' cell '
            print ('You can choose more ' + str(3 - move_index) + ' ' + text_cells + '. Press
enter to no longer choose ')
            if len(move) == 0:
                return None
```

```
        move = input()
    return int(move)
def chooseRandomMoveFromList(board, moveList):
    possibleMoves = []
    for i in moveList:
        if isSpacefree(board, i):
            possibleMoves.append(i)
        if len(possibleMoves) !=0:
            return random.choice(possibleMoves)
        else:
            return None
def getComputerMove(board, computerLetter):
    if computerLetter == 'X':
        playerLetter = 'O'
    else:
        playerLetter = 'X'
    for i in range(1,17):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, computerLetter, i)
            return i
    raise Exception("Board is full")
def isBoardFull(board):
    for i in range(1, 17):
        if isSpaceFree(board, i):
            return False
    return True
while True:
    theBoard = [' ']*17
    playerLetter, computerLetter = inputPlayerLetter()
    turn = whoGoesFirst()
    print('  ' + turn + ' will move first ')
    gameIsPlaying = True
    while gameIsPlaying:
        if turn == 'player':
            drawBoard(theBoard)
            prev_moves = []
            for i in range(3):
                move = getPlayerMove(theBoard, i, prev_moves)
                if move is None:
                    turn = 'computer'
                    break
                makeMove(theBoard, playerLetter, move)
                prev_moves.append(move)
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('You win')
                    gameIsPlaying = False
                else:
                    turn = 'computer'
        else:
            move = getComputerMove(theBoard, computerLetter)
            makeMove(theBoard, computerLetter, move)
            column_or_row = random.choice(['column', 'row'])
            amount = random.randint(0, 3)
            for i in range(amount):
                if column_or_row == 'row':
                    new_move = move + i+1
                    if (new_move-1) // 3 != (move-1) // 3:
                        break
                else:
                    new_move = move + 3*(i+1)
                    print('nm: ' + str(new_move) + ' ' + str(move))
                    if new_move < 0 or new_move > 16:
                        break
                if isSpaceFree(theBoard, new_move):
                    makeMove(theBoard, computerLetter, new_move)
                else:
                    break
            if isBoardFull(theBoard):
                drawBoard(theBoard)
                print('You have lost')
                gameIsPlaying = False
            else:
                turn = 'player'
    if not playAgain():
        break
```

References

1. Bouton, C.L. Nim, a Game with a Complete Mathematical Theory. *The Annals of Mathematics* **1901-1902**, 2nd Ser., Vol. 3, No. 1/4. pp. 35-39.
2. Siegel, A.N. *Combinatorial Game Theory*; American Mathematical Society: Providence, USA, 2013.
3. El-Seidy, E.; Hussein, S.E.S.; Alabdala, A.T. Models of Combinatorial Games and Some Applications: A Survey. *Journal of Game Theory* **2016**, Vol. 5(2), pp. 27-41.
4. Oltean, M. Evolving Winning Strategies for Nim-like Games, In *IFIP Student Forum*, 2004, pp. 353–364.

5. Iacono, J., Mazur, K. Tactix on an S-shaped Board. Proceedings of 22nd Annual Fall Workshop on Computational Geometry, College Park MD, 2012, November, 9-10, pp. 17-18.
6. Ooya, T.; Akiyama, J. Impact of binary and Fibonacci expansions of numbers on winning strategies for Nim-like games, *International Journal of Mathematical Education in Science and Technology* **2003**, Vol. 34, No. 1, pp. 121-128.
7. Albert, M.H.; Nowakowski, R.J.; Wolfe, D. *Lessons in play: an introduction to combinatorial game theory*, 2nd ed.; CRC Press: Boca Raton, Florida:, USA, 2019.
8. Krykun, I.H. Large deviation principle for stochastic equations with local time. *Theory of Stochastic Processes* **2009**. *15(31), No. 2.* pp. 140–155.
9. Krykun, I.H. Functional law of the iterated logarithm type for a skew Brownian motion. *Teorija Imovirnostej ta Matematychna Statystyka* **2012**, *87*. pp. 60-77 (*in Ukrainian*)
10. Krykun, I.H.; Makhno, S.Ya. The Peano phenomenon for Itô equations. *Journal of Mathematical Sciences* **2013**, *192, Issue 4*, pp. 441–458. DOI: 10.1007/s10958-013-1407-5
11. Krykun, I.H. Functional law of the iterated logarithm type for a skew Brownian motion. *Theory of Probability and Mathematical Statistics* **2013**, *87*, pp. 79–98. DOI: 10.1090/S0094-9000-2014-00906-0
12. Krykun, I.H. Convergence of skew Brownian motions with local times at several points that are contracted into a single one. *Journal of Mathematical Sciences* **2017**, *221, Issue 5*, pp. 671–678. DOI: 10.1007/s10958-017-3258-y
13. Krykun, I.H. The Arc-Sine Laws for the Skew Brownian Motion and Their Interpretation. *Journal of Applied Mathematics and Physics* **2018**, *6, No. 2*, pp. 347–357. DOI: 10.4236/jamp.2018.62033
14. Krykun, I. H. The Arctangent Regression and the Estimation of Parameters of the Cauchy Distribution. *Journal of Mathematical Sciences* **2020**, *249, Issue 5*, pp. 739-753.
15. Krykun, I.H. The arcsine laws in the modelling of the natural processes depending on random factors. In *Physical and mathematical justification of scientific achievements: collective monograph*, Primedia eLaunch LLC: Boston, USA, **2020**; pp. 24-33. DOI: 10.46299/ISG.2020.MONO.PHYSICAL.III
16. Krykun, I.H. New Approach to Statistical Analysis of Election Results. *International Journal of Mathematical, Engineering, Biological and Applied Computing* **2022**, *1, No. 2*, pp. 68–76. DOI: 10.31586/ijmebac.2022.466
17. Knorps, M. NIM and combinatorial games on graphs. Faculty of Applied Physics and Mathematics, Gdańsk University of Technology, 2007.
18. Milvang-Jensen, B.C.A. *Combinatorial games, theory and applications*, 2000.