УДК 044 DOI: 10.18698/2541-8009-2019-05-482

# АЛГОРИТМ «МИНИМАКС» И ЕГО РЕАЛИЗАЦИЯ НА ПРИМЕРЕ ИГРЫ «КРЕСТИКИ-НОЛИКИ»

**Н.А. Жданов** nzhdanov@forkode.ru
SPIN-код: 9703-6770

**Ю.М. Бурханова** ybourkhanova@gmail.com
SPIN-код: 8118-8362

**Ю.О. Воронецкий** z3xxjv@gmail.com
SPIN-код: 2780-0081

**МГТУ им. Н.Э. Баумана, Москва, Российская Федерация**

**Аннотация**

*В настоящее время машинное обучение представляет собой актуальное и стремительно развивающееся направление в области информационных технологий, поскольку нынешнее аппаратное обеспечение обладает высокой эффективностью, необходимой для решения подобных задач. Цель машинного обучения — создание искусственного интеллекта, способного самостоятельно принимать решения, но на данном этапе развития технологии главное — автоматизировать процесс принятия решений человеком. В рамках данной статьи исследован и реализован алгоритм, способный принимать оптимальные решения для выбора хода в игре «крестики-нолики», а также представлен метод оптимизации полученного решения. Результаты работы алгоритма предоставлены и проанализированы.*

**Ключевые слова**

*Алгоритм «минимакс», метод «альфа-бета отсечение», рекурсия, оценочная функция, Python, «крестики-нолики», максимизация, минимизация, метод ветвей и границ*

Поступила в редакцию 07.05.2019


**Ключевые идеи алгоритма.** Алгоритм «минимакс» относится к классу эвристических алгоритмов, позволяющих искусственному интеллекту принимать решения в играх. Данный алгоритм является рекурсивным, вызывая подзадачи до тех пор, пока очередная не выполнится без дополнительных вызовов [1].

Ключевая идея алгоритма применительно к играм состоит в поиске наилучшего хода. На рис. 1 изображена ситуация, в которой игра в «крестики-нолики» близится к завершению, а ход предстоит сделать игроку X. Алгоритм рассматривает три доступных хода, построив тем самым дерево рекурсии. Ветвь считается законченной, если ее листовой узел выполняет условия окончания игры, данное условие является «крайним случаем» рекурсии.

Таким образом, условие окончания игры выполняется, если метка одного из игроков последовательно занимает три клетки по горизонтали, вертикали или диагонали либо игровое поле не содержит свободных клеток. В данном случае вершиной дерева является состояние игровой доски, которую требуется заполнить в контексте алгоритма единственным ходом игрока X, называемым

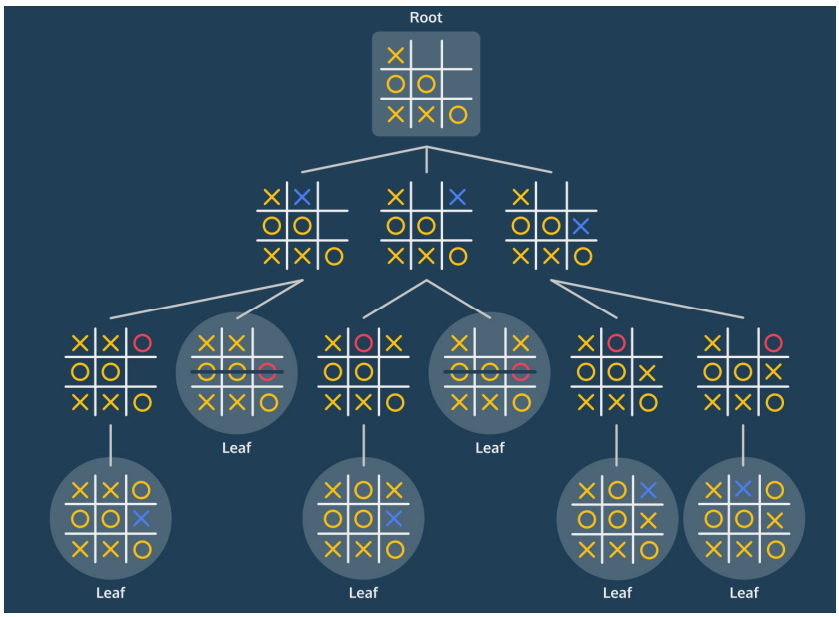


**Рис. 1.** Дерево принятия решений алгоритма

максимизирующим. Каждый из возможных вариантов порождает новые гипотетические ситуации, результат которых зависит от хода игрока O, называемого минимизирующим. Для принятия оптимального решения алгоритму необходимо оценивать возможные ходы [2]. Описав условия окончания игры и определив ключевые понятия, оптимальной оценкой победы максимизирующего игрока следует задать значение, равное единице, минимизирующего — обратной величиной, а случай ничьей — нулем. Оценивание и возврат результатов происходит снизу вверх, т. е. от «листьев» к «корню», причем максимизирующий игрок ожидает получить максимально возможную оценку, а минимизирующий — наоборот, как показано на рис. 2.

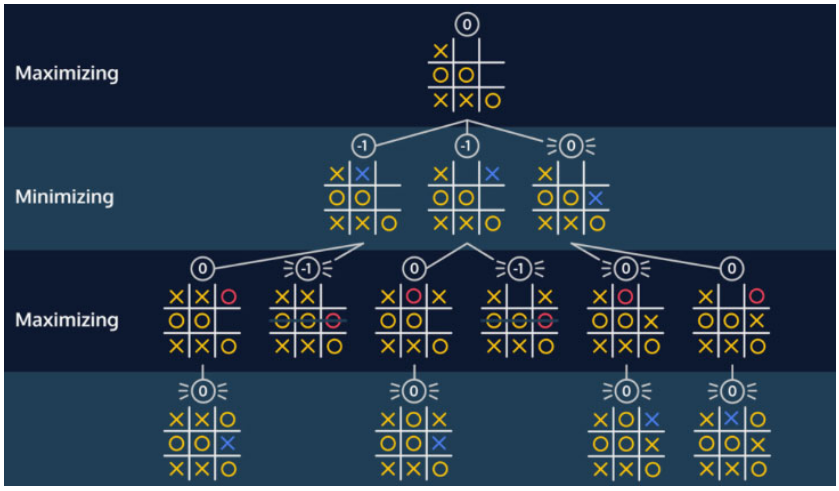


**Рис. 2.** Присвоение и возврат оценки

Таким образом, вызывающая программа — «корень дерева» — получает наилучшую оценку с учетом того, что ход соперника был оптимальным. Результатом работы алгоритма является выбор хода, обеспечивающего оптимальный результат, в контексте ситуации, изображенной на рис. 2, следует, что наиболее оптимистичным исходом максимизирующего игрока является ничья, поскольку оценочная функция вернула нулевое значение, а значит, следует выбрать ход, гарантированно обеспечивающий этот исход [3].

```
""" Вызов данной выведет на экран конкретную игровую доску"""
def print_board(board):
    print("|-------------|")
    print("| Tic Tac Toe |")
    print("|-------------|")
    print("|             |")
    print("|     " + board[0][0] + " " + board[0][1] + " " + board[0][2] + "     |")
    print("|     " + board[1][0] + " " + board[1][1] + " " + board[1][2] + "     |")
    print("|     " + board[2][0] + " " + board[2][1] + " " + board[2][2] + "     |")
    print("|             |")
    print("|-------------|")
    print()

""" Вызов данной функции означает выполнение хода игроком на конкретной игровой доске"""
def select_space(board, move, turn):
    if move not in range(1,10):
        return False
    row = int((move-1)/3)
    col = (move-1)%3
    if board[row][col] != "X" and board[row][col] != "O":
        board[row][col] = turn
        return True
    else:
        return False

""" Вызов данной выведет доступные ходы для конкретного игрового поля"""
def available_moves(board):
    moves = []
    for row in board:
        for col in row:
            if col != "X" and col != "O":
                moves.append(int(col))
    return moves

""" Вызов данной вернет True если победил один из игроков, False иначе"""
def has_won(board, player):
    for row in board:
        if row.count(player) == 3:
            return True
    for i in range(3):
        if board[0][i] == player and board[1][i] == player and board[2][i] == player:
            return True
    if board[0][0] == player and board[1][1] == player and board[2][2] == player:
        return True
    if board[0][2] == player and board[1][1] == player and board[2][0] == player:
        return True
    return False
```

**Рис. 3.** Реализация интерфейсов взаимодействия с игровым полем

**Оценка эффективности алгоритма.** К позитивным сторонам данного алгоритма следует отнести адекватную модель принятия решений, которая в рассматриваемом примере обеспечивает вызывающему игроку победу или ничью, как будет доказано ниже. Данный эффект достигается благодаря тому, что алгоритму заранее известны все возможные исходы партии уже в самом начале игры [4]. Однако алгоритм ограничен в реализации по ряду причин. Во-первых, очевидным является факт, что рассмотрение каждой гипотезы влечет за собой создание копии игровой доски, а это говорит о высокой ресурсоемкости алгоритма, т. е. эффективность алгоритма по памяти весьма низкая. То же касается и скорости выполнения: при линейном росте размера игрового поля число возможных решений возрастает экспоненциально, а поскольку прием рекурсии использует стек

вызовов, имеющий ограниченный ресурс, выполнения программы может не произойти вовсе. Далее кратко опишем метод, способный оптимизировать выполнение алгоритма и ликвидировать описанную проблему при ситуации, когда размер игрового поля превышает рассмотренный в данном примере [5].

**Реализация алгоритма с помощью средств языка Python.** Первым этапом реализации является создание игрового поля и интерфейсов взаимодействия с ним. Исходный код, реализующий описанное, представлен на рис. 3.

```
"""Данная функция возвращает True если 1 из игроков победил, либо на игровом поле не осталось мест для хода, иначе возвращает False"""
def game_is_over(board):
  if has_won(board, 'X') or has_won(board, '0') or not available_moves(board):
    return True
  return False
"""Данная функция возращает оценку при достижении "крайнего" случая"""
def evaluate_board(board):
  if has_won(board, 'X'):
    return 1
  if has_won(board, '0'):
    return -1
  else:
    return 0
```

**Рис. 4.** «Крайний случай» рекурсии и оценочная функция

```
def minimax(input_board, is_maximizing):
  """Крайний" случай рекурсии - игра окончена"""
  if game_is_over(input_board):
    return [evaluate_board(input_board), ""]
  """Инициализируем значения best_value и best_move"""
  best_move = ""
  """Случай, когда ход максимизирующего игрока - "X" """
  if is_maximizing == True:
    best_value = -float("Inf")
    symbol = "X"
  """Случай минимизирующего игрока - "0" """
  else:
    best_value = float("Inf")
    symbol = "0"
  """Пройдем циклом по всем возмодным ходам, для того чтобы выбрать наилучший
  путем рекурсивных вызовов функции minimax с копией игровой доски.
  Как только рекурсия достигнет "крайнего" случая, она вернет значения из [0, 1, -1]
  для функции, которая ее вызвала, до тех пор, пока самая "верхняя" функция в стеке
  вызов(minimax с текущей игровой доской) не получит свое значение"""
  for move in available_moves(input_board):
    new_board = deepcopy(input_board)
    select_space(new_board, move, symbol)
    hypothetical_value = minimax(new_board, not is_maximizing)[0]
    if is_maximizing == True and hypothetical_value > best_value:
      best_value = hypothetical_value
      best_move = move
    if is_maximizing == False and hypothetical_value < best_value:
      best_value = hypothetical_value
      best_move = move
  return [best_value, best_move]
```

**Рис. 5.** Функция, реализующая поведение алгоритма «минимакс»

На рис. 6 продемонстрирован полученный результат. Игрок X — автор, а его соперник — искусственный интеллект, использующий реализованный алгоритм, для краткости описание промежуточного процесса опущено [8]. В итоге при многократных запусках, действуя оптимальным образом, алгоритм всегда сводил партию к ничьей, а при совершении человеком ошибки — выигрывал.

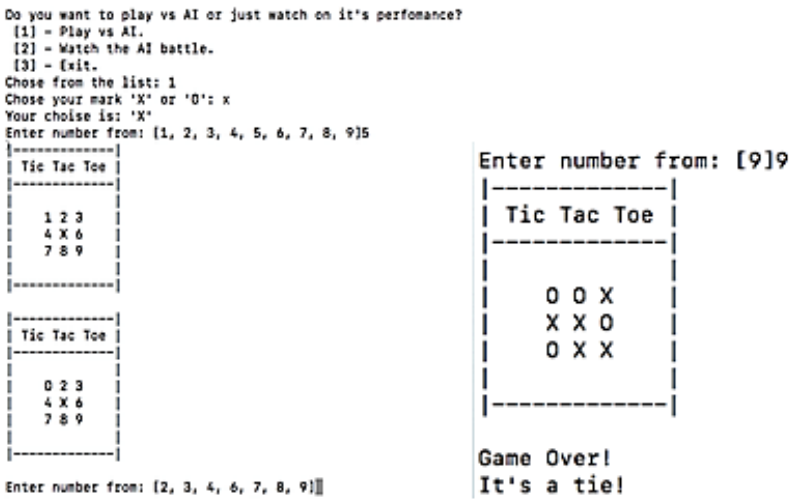


**Рис. 6.** Демонстрация результата оптимизации алгоритма при большей входной размерности

**Метод оптимизации алгоритма.** Как описано ранее, существует метод, способный оптимизировать работу алгоритма при больших размерах игрового поля. Метод имеет название «альфа-бета отсечение», а по своей сути он является методом «ветвей и границ» [9]. Основная его идея заключается в том, чтобы после «прохода» по одной из ветвей дерева решений «отсекать» ветви, заведомо не имеющие оптимального решения, относительно коэффициентов альфа и бета, полученных на первом проходе. Альфа — это минимально возможная оценка, которую может получить максимизирующий игрок, инициализируется значением минус бесконечности и наоборот. Если в одном из узлов ветви значения альфа больше либо равно бета, найден ход, который гарантирует победу максимизирующему игроку. Если значения этих параметров в текущем узле не улучшаются, ветвь не следует рассматривать, поскольку все ее «потомки», как и она сама, не содержат оптимального решения [10]. Таким образом, описанный метод позволяет находить оптимальное решение, заметно сократив при этом число рекурсивных вызовов.

## Литература

[1] Pollice G., Selkow S., Heineman G.T. Algorithms in a nutshell. O'Reilly Media, 2008.
[2] Tadelis S. Game theory: an introduction. Princeton University Press, 2013.
[3] Tic Tac Toe: understanding the Minimax algorithm. *neverstopbuilding.com*: веб-сайт. URL: https://www.neverstopbuilding.com/blog/minimax (дата обращения: 10.02.2019).
[4] Jain Sh. The ultimate beginners guide to analysis of algorithm. *codeburst.io: веб-сайт*. URL: https://codeburst.io/the-ultimate-beginners-guide-to-analysis-of-algorithm-b8d32aa909c5 (дата обращения: 10.02.2019).
[5] Turney K. Recursion is not hard: a step-by-step walkthrough of this useful programming technique. *medium.freecodecamp.org: веб-сайт*. URL: https://medium.freecodecamp.org/recursion-is-not-hard-858a48830d83 (дата обращения: 05.02.2019).

[6] Introduction to Minimax algorithm. *baeldung.com: веб-сайт*.
URL: https://www.baeldung.com/java-minimax-algorithm (дата обращения: 08.02.2019).
[7] Lutz M. Programming Python. O'Reilly Media, 2011.
[8] Lutz M. Learning Python: powerful object-oriented programming. O'Reilly Media, 2013.
[9] Jain R. Minimax algorithm with Alpha-beta pruning. *www.hackerearth.com: веб-сайт*.
URL: https://www.hackerearth.com/blog/artificial-intelligence/minimax-algorithm-alpha-beta-pruning/ (дата обращения: 14.02.2019).
[10] Alpha-Beta. *chessprogramming.org: веб-сайт*.
URL: https://www.chessprogramming.org/Alpha-Beta (дата обращения: 18.02.2019).

**Жданов Никита Алексеевич** — студент кафедры «Компьютерные системы и сети», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

**Бурханова Юлия Мансуровна** — студентка кафедры «Компьютерные системы и сети», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

**Воронецкий Юлиан Олегович** — студент кафедры «Компьютерные системы и сети», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

**Научный руководитель** — Фомин Михаил Михайлович, преподаватель кафедры «Компьютерные системы и сети», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

# THE MINIMAX ALGORITHM AND ITS IMPLEMENTATION ON THE EXAMPLE OF THE NOUGHTS-AND-CROSSES GAME

**N.A. Zhdanov** nzhdanov@forkode.ru
SPIN-code: 9703-6770

**Yu.M. Burkhanova** ybourkhanova@gmail.com
SPIN-code: 8118-8362

**Yu.O. Voronetskiy** z3xxjv@gmail.com
SPIN-code: 2780-0081

**Bauman Moscow State Technical University, Moscow, Russian Federation**

**Abstract**

*At present, machine learning is an actual and rapidly developing area in the field of information technologies, since the current hardware is highly efficient for solving such problems. The goal of machine learning is the creation of artificial intelligence capable of making decisions independently, but at this stage of technology development the main thing is to automate the decision-making process by humans. In this article, we investigated and implemented an algorithm capable of making optimal decisions for the choice of a move in a noughts-and-crosses game, and also presented a method for optimizing the solution obtained. The results of the algorithm are provided and analyzed.*

**Keywords**

*Minimax algorithm, alpha-beta clipping, recursion, evaluation function, Python, noughts-and-crosses, maximization, minimization, branch and bound method*

Received 07.05.2019



## References

[1] Pollice G., Selkow S., Heineman G.T. Algorithms in a nutshell. O'Reilly Media, 2008.
[2] Tadelis S. Game theory: an introduction. Princeton University Press, 2013.
[3] Tic Tac Toe: understanding the Minimax algorithm. *neverstopbuilding.com: website.* URL: https://www.neverstopbuilding.com/blog/minimax (accessed: 10.02.2019).
[4] Jain Sh. The ultimate beginners guide to analysis of algorithm. *codeburst.io: website.* URL: https://codeburst.io/the-ultimate-beginners-guide-to-analysis-of-algorithm-b8d32aa909c5 (accessed: 10.02.2019).
[5] Turney K. Recursion is not hard: a step-by-step walkthrough of this useful programming technique. *medium.freecodecamp.org: website*. URL: https://medium.freecodecamp.org/recursion-is-not-hard-858a48830d83 (accessed: 05.02.2019).
[6] Introduction to Minimax algorithm. *baeldung.com: website*. URL: https://www.baeldung.com/java-minimax-algorithm (accessed: 08.02.2019).
[7] Lutz M. Programming Python. O'Reilly Media, 2011.
[8] Lutz M. Learning Python: powerful object-oriented programming. O'Reilly Media, 2013.
[9] Jain R. Minimax algorithm with Alpha-beta pruning. *www.hackerearth.com*: *website*. URL: https://www.hackerearth.com/blog/artificial-intelligence/minimax-algorithm-alpha-beta-pruning/ (accessed: 14.02.2019).
[10] Alpha-Beta. *chessprogramming.org: website*. URL: https://www.chessprogramming.org/Alpha-Beta (accessed: 18.02.2019).

**Zhdanov N.A.** — Student, Department of Computer Systems and Networks, Bauman Moscow State Technical University, Moscow, Russian Federation.

**Burkhanova Yu.M.** — Student, Department of Computer Systems and Networks, Bauman Moscow State Technical University, Moscow, Russian Federation.

**Voronetskiy Yu.O.** — Student, Department of Computer Systems and Networks, Bauman Moscow State Technical University, Moscow, Russian Federation.

**Scientific advisor** — Fomin M.M., Assistant Professor, Department of Computer Systems and Networks, Bauman Moscow State Technical University, Moscow, Russian Federation.