МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВЕРО-ВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ ИМЕНИ М.К.

АММОСОВА»

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАТИКИ

**Отчёт по учебной ознакомительной практике**

по дисциплине «Математическое моделирование и вычислительная математика» направления 01.03.02 «Прикладная математика и информатика»

студента группы ИИ-ПМИ-23 Блахирова Тимура Викторовича / \_\_\_\_\_\_\_\_\_ (подпись) Руководитель практики: доцент, к.т.н. Акимов М.П.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_

(оценка / подпись)

Якутск 2025

## СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc210006117)

[ОСНОВНАЯ ЧАСТЬ 3](#_Toc210006118)

[ЗАКЛЮЧЕНИЕ 8](#_Toc210006119)

[ПРИЛОЖЕНИЯ 9](#_Toc210006120)

# ВВЕДЕНИЕ

Тему для проекта по учебной практике нам студентам дали возможность выбрать самим, что ближе лежит, главное чтобы тема была связана с IT, для достижения целей надо было пройти онлайн-курсы, а так же выложить все файлы связанные с практикой на платформу GitHub.

Я выбрал мини-проект «Крестики-нолики»: это приложение – игра основанный на языке Python 3.9.3

В приложении можно попытаться переиграть компьютера, в другую очередь компьютер будет делать все, чтобы не проиграть вам.

В приложении будет как консольный вид, так и графический, и там и там, при входе будет видна доска с размером 3x3, где будут происходить бои.

Я выбрал именно этот проект, потому что я с детства интересуюсь играми, и мне стало интересно сможет ли выиграть человек который знает все шаблоны ходов для победы, или же компьютер.

# ОСНОВНАЯ ЧАСТЬ

1. **Подготовительный этап**

Прежде всего я прошел онлайн-курс на Stepik под названием “Машинное обучение”(Рисунок 1), а также “Осознанный Python - База”(Рисунок 2). В ходе данных курсов я узнал принцип работы данного машинного обучения и освежил свою память насчет Python.



Рисунок 1. Сертификат об прохождении курса Stepik

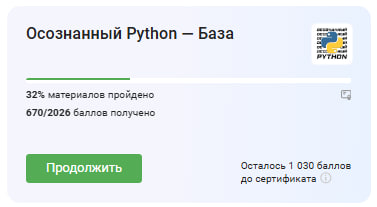


Рисунок 2. Прогресс прохождения курса Stepik

**1.1. Мозговой штурм**

Далее, чтобы реализовать свой проект, мне надо было продумать логику игры и примерный вид кода, стеков/библиотек.

**2. Изучение технологий и инструментов**

**2.1 Основные технологии**

Программа была написана на Python с использованием стандартных библиотек и дополнительного модуля termcolor для цветового оформления. Основной стек технологий:

* Python 3.x - основной язык программирования
* random - для реализации случайных ходов ИИ
* termcolor - для цветового оформления вывода в консоль

**2.2 Начало начал**

Был выбран объектно-ориентированный подход с одним основным классом Game, что позволяет четко разделить ответственность и делает код легко расширяемым.

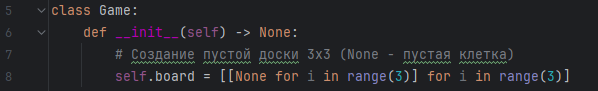


Рисунок 3. Фрагмент кода. Класс Game

**3.3 Разработка игровой логики**

**3.1 Проектирование структуры приложения**

Проект представляет из себя консольную игру "Крестики-нолики" с интеллектуальным ИИ. Архитектура построена по объектно-ориентированному принципу с четким разделением ответственности между компонентами.

**Основные компоненты системы:**

* Класс Game - управление игровым состоянием
* Функция check\_win - проверка условий победы
* Функция computerMove - искусственный интеллект
* Функция play - обработка ходов игроков

**3.2 Реализация игрового поля**

Было решено сделать игровое поле в виде двумерного массива размером 3x3.

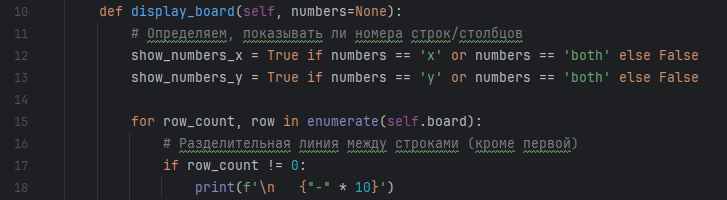


Рисунок 4. Фрагмент кода. Функция display\_board

Этот метод обеспечивает гибкое отображение игрового поля с возможностью показа номеров строк и столбцов для удобства пользователя.

**3.3. Механизм проверки победы**

Реализована комплексная проверка всех возможных выигрышных комбинаций

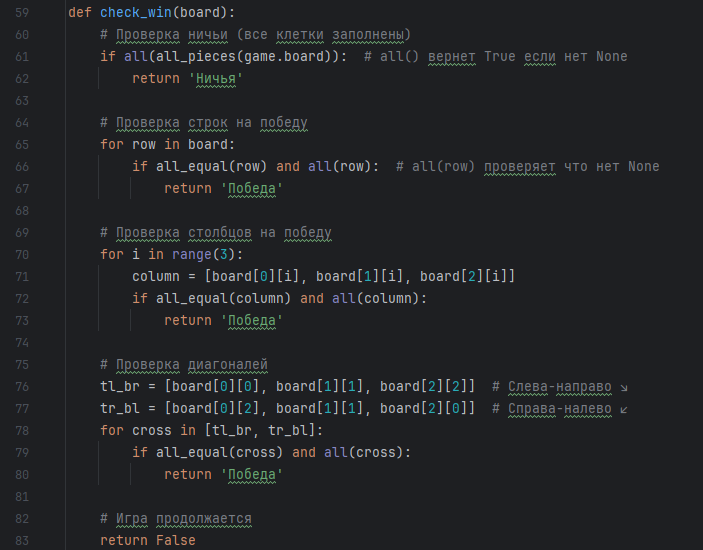


Рисунок 5. Фрагмент кода. Функция check\_win

**4. Разработка интеллектуального ИИ**

**4.1. Проблема создания умного противника**

Первоначально я решил сделать пустую болванку, тоесть сделал чтобы он просто делал случайные ходы

**4.2. Алгоритм искусственного интеллекта**

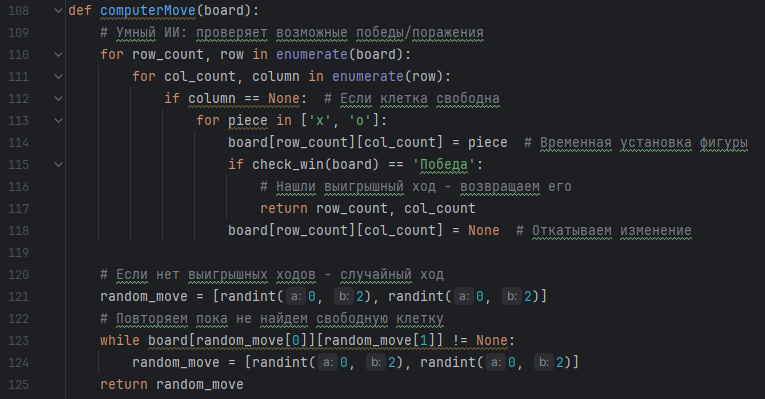
Путем множество проб и ошибок был реализован двухуровневый алгоритм с приоритетами 

Рисунок 6. Фрагмент кода. Функция computeMove

Стратегия данного ИИ включает в себя:

1. Приоритет победы - проверка, может ли компьютер выиграть следующим ходом
2. Блокировка противника - предотвращение выигрышных ходов человека
3. Случайный ход - выбор случайной свободной клетки при отсутствии критических ситуаций

**5. Пользовательский интерфейс и взаимодействие**

**5.1. Система ввода и валидации**

Для обработки пользовательского ввода реализована рекурсивная функция, обеспечивающая повторный запрос при некорректном вводе

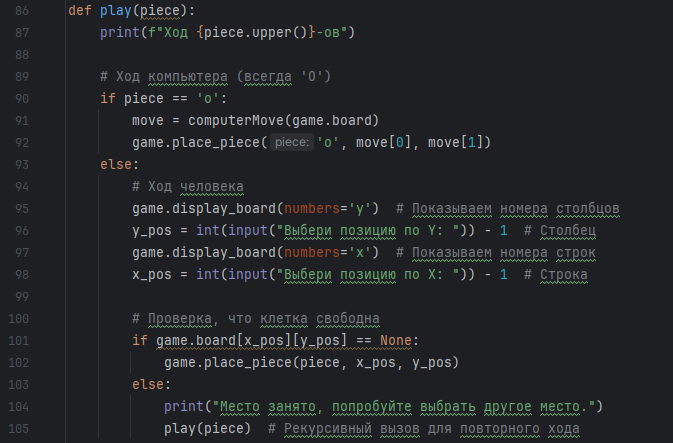


Рисунок 7. Фрагмент кода. Функция play

**6. Основной игровой цикл**

Дальнейшим этапом было реализовать основной игровой цикл с поочередными ходами игроков и проверкой условий окончания игры



Рисунок 8. Фрагмент кода.

**8. Дополнительные функции и возможности**

Так как код успешно работает, при проверке, я проиграл ИИ и захотел взять реванш, чтобы не перезапускать каждый раз код была сделана функция перезапуска системы

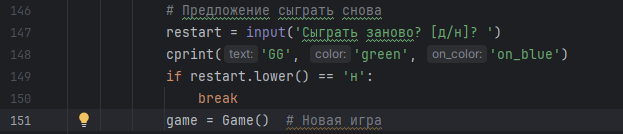


Рисунок 9. Фрагмент кода.

А также для оптимальной и стабильной работы кода были добавлены вспомогательные функции.

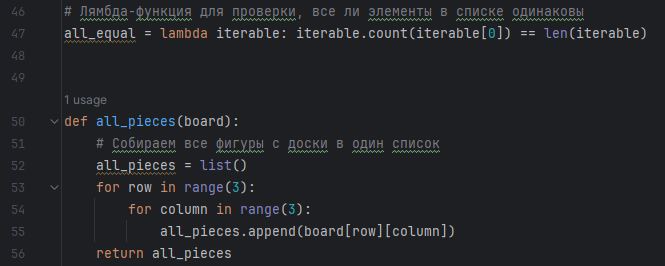


Рисунок 9. Фрагмент кода. Вспомогательные функции

# ЗАКЛЮЧЕНИЕ

Итоги этой работы была реализована игра “Крестики-нолики”, в котором можно:

* Играть против компьютера у которого есть своя логика;
* Делать ходы по координатам, выбирая позицию на поле;
* Наблюдать за стратегией компьютера, который пытается выиграть или блокировать ходы игрока;
* Определять победителя;

В ходе работы и подготовки, я подкрепил свои знания по машинному обучению, а также освежил свои навыки программирования на Python, прошел онлайн - курс на платформе Stepik, разработал консольное приложение\*, где полностью работают функции и просто хорошо провел время решая проблемы кода.

# ПРИЛОЖЕНИЯ

# crest-noliki.py

**from** **random** **import** randint

**from** **termcolor** **import** colored, cprint

**class** **Game**:

**def** **\_\_init\_\_**(self) -> **None**:

# Создание пустой доски 3x3 (None - пустая клетка)

self.board = [[**None** **for** i **in** range(**3**)] **for** i **in** range(**3**)]

**def** **display\_board**(self, numbers=**None**):

# Определяем, показывать ли номера строк/столбцов

show\_numbers\_x = **True** **if** numbers == 'x' **or** numbers == 'both' **else** **False**

show\_numbers\_y = **True** **if** numbers == 'y' **or** numbers == 'both' **else** **False**

**for** row\_count, row **in** enumerate(self.board):

# Разделительная линия между строками (кроме первой)

**if** row\_count != **0**:

print(f'**\n** {"-" \* **10**}')

**for** col\_count, col **in** enumerate(row):

# Показываем номера строк слева

**if** col\_count == **0** **and** show\_numbers\_x:

print(str(row\_count + **1**), ' ', end="")

**elif** col\_count == **0**:

print(' ', end="")

# Заменяем None на пробел для красивого отображения

col = ' ' **if** col **is** **None** **else** col

print(col, end="")

# Вертикальные разделители между столбцами

**if** col\_count != **2**:

print(" | ", end="")

# После последней строки добавляем номера столбцов

**if** row\_count == **2**:

**if** show\_numbers\_y:

print("**\n\n** 1 2 3", end="")

print('**\n**')

**def** **place\_piece**(self, piece, x\_pos, y\_pos):

# Размещение фигуры на доске

self.board[x\_pos][y\_pos] = piece

# Лямбда-функция для проверки, все ли элементы в списке одинаковы

all\_equal = **lambda** iterable: iterable.count(iterable[**0**]) == len(iterable)

**def** **all\_pieces**(board):

# Собираем все фигуры с доски в один список

all\_pieces = list()

**for** row **in** range(**3**):

**for** column **in** range(**3**):

all\_pieces.append(board[row][column])

**return** all\_pieces

**def** **check\_win**(board):

# Проверка ничьи (все клетки заполнены)

**if** all(all\_pieces(game.board)): # all() вернет True если нет None

**return** 'Ничья'

# Проверка строк на победу

**for** row **in** board:

**if** all\_equal(row) **and** all(row): # all(row) проверяет что нет None

**return** 'Победа'

# Проверка столбцов на победу

**for** i **in** range(**3**):

column = [board[**0**][i], board[**1**][i], board[**2**][i]]

**if** all\_equal(column) **and** all(column):

**return** 'Победа'

# Проверка диагоналей

tl\_br = [board[**0**][**0**], board[**1**][**1**], board[**2**][**2**]] # Слева-направо ↘

tr\_bl = [board[**0**][**2**], board[**1**][**1**], board[**2**][**0**]] # Справа-налево ↙

**for** cross **in** [tl\_br, tr\_bl]:

**if** all\_equal(cross) **and** all(cross):

**return** 'Победа'

# Игра продолжается

**return** **False**

**def** **play**(piece):

print(f"Ход {piece.upper()}-ов")

# Ход компьютера (всегда 'O')

**if** piece == 'o':

move = computerMove(game.board)

game.place\_piece('o', move[**0**], move[**1**])

**else**:

# Ход человека

game.display\_board(numbers='y') # Показываем номера столбцов

y\_pos = int(input("Выбери позицию по Y: ")) - **1** # Столбец

game.display\_board(numbers='x') # Показываем номера строк

x\_pos = int(input("Выбери позицию по X: ")) - **1** # Строка

# Проверка, что клетка свободна

**if** game.board[x\_pos][y\_pos] == **None**:

game.place\_piece(piece, x\_pos, y\_pos)

**else**:

print("Место занято, попробуйте выбрать другое место.")

play(piece) # Рекурсивный вызов для повторного хода

**def** **computerMove**(board):

# Умный ИИ: проверяет возможные победы/поражения

**for** row\_count, row **in** enumerate(board):

**for** col\_count, column **in** enumerate(row):

**if** column == **None**: # Если клетка свободна

**for** piece **in** ['x', 'o']:

board[row\_count][col\_count] = piece # Временная установка фигуры

**if** check\_win(board) == 'Победа':

# Нашли выигрышный ход - возвращаем его

**return** row\_count, col\_count

board[row\_count][col\_count] = **None** # Откатываем изменение

# Если нет выигрышных ходов - случайный ход

random\_move = [randint(**0**, **2**), randint(**0**, **2**)]

# Повторяем пока не найдем свободную клетку

**while** board[random\_move[**0**]][random\_move[**1**]] != **None**:

random\_move = [randint(**0**, **2**), randint(**0**, **2**)]

**return** random\_move

game = Game()

**while** **True**:

**for** piece **in** ['x', 'o']: # Поочередные ходы

play(piece)

game.display\_board()

result = check\_win(game.board)

# Проверка окончания игры

**if** result == 'Победа' **or** result == 'Ничья':

game.display\_board()

print('-' \* **20**)

**if** result == 'Победа':

print(piece.upper(), 'Победил!') # Текущий игрок победил

**else**:

print('Ничья')

print('-' \* **20**)

# Предложение сыграть снова

restart = input('Сыграть заново? [д/н]? ')

cprint('GG', 'green', 'on\_blue')

**if** restart.lower() == 'н':

**break**

game = Game() # Новая игра

## Ссылки

<https://github.com/svyyyx/SummerPractice> - ссылка на репозиторий GitHub для ознакомления с проектом самостоятельно

**Использованные онлайн-ресурсы**

* stepik.org
* youtube.org
* python.org
* stackoverflow.com