МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

на лабораторную работу №10

по дисциплине «Алгоритмы и структуры данных»

|  |  |
| --- | --- |
| **Подп.и** |  |
| **Инв** |  |
| **Вза** |  |
| **Подп.и** |  |
| **Инв** |  |

**Компьютерная игра «крестики-нолики».**

Р.02069337. №23/737-Вариант 22 ПЗ-

Листов <12>

Исполнитель:

студент гр. ИСТбд-23

Селезнев К. В.

«\_\_» \_\_\_\_\_\_\_\_ 2025 г.

2025

**Введение**

*Наименование и условное обозначение разрабатываемого приложения:*

Программное приложение *"*Крестики-нолики" (Tic-Tac-Toe)

*Наименования реализуемой игры:*

Классическая игра "Крестики-нолики".

*Описание и обоснование подхода:*

При создании приложения был применен объектно-ориентированный подход. Для формирования графического интерфейса использовались базовые компоненты библиотеки Tkinter. Искусственный интеллект противника реализован на основе алгоритма минимакса, который является оптимальным решением для поиска выигрышных стратегий в играх с полной информацией и нулевой суммой, к категории которых относится "Крестики-нолики".

*Краткое описание реализованного приложения:*

Разработанное приложение представляет собой интерактивную игровую платформу для двух участников (пользователь и компьютер), взаимодействующих на квадратном поле размером 3×3 клетки. Участники поочередно размещают свои маркеры ("X" или "O") на свободных клетках игрового поля. Победа присуждается игроку, первым выстроившим непрерывную линию из трех своих символов по горизонтали, вертикали или диагонали. В случае заполнения всех клеток поля без образования выигрышной комбинации фиксируется ничейный результат.

1. **Проектная часть**
   1. Постановка задачи на разработку приложения

Подробное описание требований и условий разработки представлено в техническом задании.

* 1. **Математические методы**

**Математическая модель: Алгоритм минимакса**

Алгоритм минимакса применяется для анализа возможных ходов и выбора оптимального решения для компьютера. Суть метода заключается в оценке каждого хода на основе прогнозируемых результатов с выбором стратегии, максимизирующей вероятность победы или минимизирующей риск поражения.

*Как работает алгоритм минимакса:*

* **Рекурсивный анализ**

Алгоритм последовательно исследует все возможные варианты ходов, начиная с текущего состояния игрового поля.

* **Оценочная функция**

Для каждого хода определяется его влияние на исход игры (победа, поражение или ничья).

* **Стратегия максимизации и минимизации**

Компьютер выбирает ход с максимальной оценкой (наибольший шанс на победу), а для игрока (человека) рассматривается ход с минимальной оценкой (снижение вероятности выигрыша оппонента).

*Обоснование выбора:*

1. **Оптимальность для игры "Крестики-нолики"**

Простота игрового поля (матрица 3×3) позволяет алгоритму полностью просчитывать все возможные комбинации ходов.

1. **Эффективность вычислений**

Для поля малого размера алгоритм работает быстро и не требует значительных вычислительных ресурсов.

1. **Прозрачность принятия решений**

Метод обеспечивает четкую и предсказуемую логику выбора хода, что гарантирует честность игрового процесса.

* 1. **Архитектура и алгоритмы**
     1. Архитектура

**Основные структуры данных:**

1. Представление игрового поля
   * В коде используется двумерный список self.board (длина 9 элементов), где каждый элемент соответствует клетке поля.
   * Значения элементов для определения очередности хода:
     + 'X' — крестик (игрок),
     + 'O' — нолик (бот).
2. Графические компоненты (Tkinter)
   * self.buttons — двумерный список кнопок (3×3), связанных с клетками поля.
     + Каждая кнопка привязана к индексу в self.board (например, кнопка [0][0] → индекс 0).
     + При нажатии вызывает метод on\_click() с передачей координат (строка, столбец).
3. Переменные состояния игры:
   * current\_player ('X' или 'O'):
     + Определяет, чей сейчас ход (изначально 'X' — игрок).
     + Меняется после каждого хода (см. метод on\_click()).

**Основные функции:**

1. \_\_init\_\_:

В приложении реализован метод \_\_init\_\_, который выполняет инициализацию игры. Он создает основное игровое поле размером 3×3 клетки и задает начальное состояние игры.

1. on\_click:

Обработка хода игрока осуществляется в методе on\_click. Этот метод принимает координаты нажатой клетки (row и col) и проверяет, свободна ли клетка и является ли текущий игрок человеком (символ 'X').

1. make\_move:

Обрабатывает клик пользователя на кнопку игрового поля. Проверяет, свободна ли кнопка, и если да, ставит символ (X или O). Если ход делал компьютер, вызывает bot\_move.

1. bot\_move:

Ход компьютера обрабатывается в методе bot\_move. Для выбора оптимального хода здесь используется алгоритм минимакс.

1. minimax:

Этот метод рекурсивно оценивает все возможные варианты развития игры. Он принимает три параметра: текущее состояние поля board, глубину рекурсии depth и флаг is\_maximizing, указывающий, чей сейчас ход (бота или игрока). Метод возвращает числовую оценку позиции: +1 за победу бота, -1 за победу игрока и 0 в случае ничьей.

1. check\_winner:

Проверка наличия победителя осуществляется в методе check\_winner. Этот метод анализирует текущее состояние игрового поля, проверяя все возможные выигрышные комбинации: 3 строки, 3 столбца и 2 диагонали. В зависимости от результата проверки метод возвращает 'X' или 'O', если обнаружен победитель, либо None, если победителя нет.

1. check\_game\_over:

Метод check\_game\_over отвечает за проверку условий окончания игры. Он использует check\_winner для определения победителя и дополнительно проверяет, полностью ли заполнено поле. В зависимости от результата проверки метод выводит соответствующее сообщение - о победе одного из игроков или о ничьей, после чего блокирует дальнейшие ходы через вызов disable\_all\_buttons.

1. make\_move:

Метод make\_move обновляет состояние игрового поля и соответствующих элементов интерфейса.

1. is\_board\_full:

Метод is\_board\_full проверяет, остались ли на поле свободные клетки.

1. disable\_all\_buttons:

Метод disable\_all\_buttons блокирует все кнопки игрового поля, предотвращая дальнейшие ходы.

1. Restart:

Метод restart полностью сбрасывает состояние игры, позволяя начать новую партию.

**Взаимодействие функций:**

1. Функция init:

При запуске программы создается игровое поле, инициализируются переменные состояния (self.board, self.current\_player) и кнопки интерфейса. Если current\_player установлен в 'O', сразу вызывается bot\_move для первого хода компьютера.

1. Взаимодействие on\_click -> make\_move -> check\_game\_over:

Когда игрок кликает по клетке, on\_click проверяет возможность хода. Если ход допустим, вызывается make\_move для обновления состояния игры, затем check\_game\_over для проверки условий завершения.

1. Функция check\_game\_over -> check\_winner:

При проверке окончания игры check\_game\_over вызывает check\_winner для анализа выигрышных комбинаций. Если победитель найден или поле заполнено, игра завершается с соответствующим сообщением.

1. Взаимодействие bot\_move -> minimax -> make\_move:

Когда ход переходит компьютеру, bot\_move анализирует возможные ходы через minimax. Получив оптимальный ход, bot\_move применяет его через make\_move, который обновляет и поле, и интерфейс.

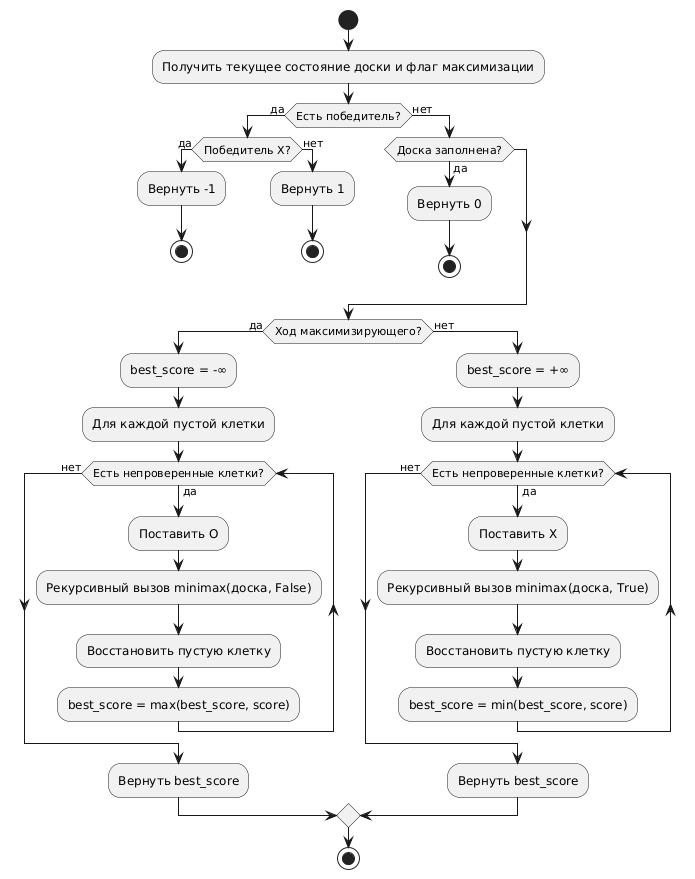
1. Функция restart:

При вызове restart происходит полный сброс игры: очищается self.board, восстанавливаются кнопки интерфейса, current\_player устанавливается в 'X'. Если компьютер должен ходить первым, сразу вызывается bot\_move.

1. Алгоритм минимакса:

В процессе работы minimax рекурсивно вызывает сам себя, а для оценки позиций использует check\_winner. Это позволяет алгоритму оценивать каждую возможную концовку игры.

* + 1. Алгоритм <Минимакс>



* 1. **Тестирование**
     1. Описание отчёта о тестировании

Тестирование проводилось для проверки работоспособности приложения, выявления ошибок и соответствия требованиям. Основой послужили test-case’ы и чек-листы.

* + 1. Цель тестирования

Цель тестирования — убедиться, что приложение работает корректно, соответствует требованиям и не содержит критических ошибок.

* + 1. Методика тестирования

Применялись методы:

1. Статическое тестирование
   * Анализ кода на:
     + Соответствие PEP-8 (стиль, отступы).
     + Логические ошибки (например, некорректные условия в check\_winner).
     + Избыточность (неиспользуемые переменные, например, turn\_count).
   * Проверка документации (пояснительной записки).
2. Ручное тестирование

Проверялись сценарии:

* Победа игрока:
  + Ручной ввод выигрышной комбинации для X.
  + Проверка, что бот блокирует выигрыш X, если это возможно.
* Победа бота:
  + Имитация "ловушки" (например, вилка 2 диагонали).
* Ничья:
  + Заполнение поля без выигрышных комбинаций.
* Ошибки интерфейса:
  + Попытка клика на занятую клетку.
  + Повторный клик после завершения игры.
    1. Проведенные тесты

1. **Тест: проверка выигрыша по строке**

* *Описание сценария:* Проверка логики определения победы при заполнении трёх клеток подряд по горизонтали.
* *Действия:*
* Игрок (X) делает ходы в клетки:
  + (0, 0) → X
  + (0, 1) → X
  + (0, 2) → X (завершающая комбинация).
* *Ожидаемый результат:*
* Программа определяет победу игрока.
* Появляется сообщение: «Победили X!».
* Все кнопки поля блокируются (state="disabled").
* Фактический результат:
* Победа корректно определяется методом check\_winner (проверка строки 0).
* Сообщение выводится через messagebox.showinfo.
* Дальнейшие ходы невозможны (кнопки неактивны).

1. **Тест: Проверка выигрыша бота**

* *Описание сценария*: Проверка, что бот (O) выигрывает при возможности.
* Действия:
  + Игрок (X) начинает игру:
    - (1, 1) → X (центр).
  + Бот (O) отвечает:
    - (0, 0) → O.
  + Игрок (X):
    - (2, 2) → X.
  + Бот (O) завершает диагональ:
    - (0, 2) → O, (2, 0) → O (победа).
* *Ожидаемый результат:*
  + Бот создаёт выигрышную комбинацию по диагонали.
  + Появляется сообщение: «Бот победил!».
* Фактический результат:
* Алгоритм minimax выбирает оптимальные ходы для победы.
* Условия завершения игры работают корректно.

1. **Тест: Ничья**

* *Описание сценария*: Проверка обработки ничьей при заполнении поля без выигрышных комбинаций.
* Действия:
* Последовательность ходов:
  + - X (0, 0), O (0, 1), X (0, 2),
    - O (1, 0), X (1, 1), O (1, 2),
    - X (2, 0), O (2, 1), X (2, 2).
* *Ожидаемый результат:*
* Появляется сообщение: «Ничья!».
* Поле полностью заполнено, дальнейшие ходы невозможны.
* Фактический результат:
* Метод is\_board\_full возвращает True.
* Игра завершается без объявления победителя.

**4. Работа алгоритма минимакса**

* *Описание сценария:* Компьютер делает оптимальный ход в сложной ситуации.
* *Действия:*

1. Поставить две «X» в одну линию, оставив третью клетку свободной.
2. Позволить компьютеру сделать ход.

* *Ожидаемый результат:*
* Компьютер блокирует победный ход человека, заполняя последнюю клетку в линии.
* Ход возвращается к человеку.
* Фактический результат:
* Человеческий игрок поставил две «X»: (0, 0) и (0, 2).
* Компьютер заблокировал победный ход, поставив «O» в (0, 1).
* Ход вернулся к человеческому игроку.

*Рекомендации по улучшению приложения:*

* Улучшение UI/UX: Добавление анимации при появлении символов на клетках. Улучшение дизайна интерфейса. Выбор знака игроком.