**1. Структуры данных**

Основные структуры данных в реализации:

**1.1. Игровое поле (self.board)**

* Представлено как двумерный список размером 3x3.
* Каждая ячейка принимает одно из следующих значений:
  + "" — пустая клетка.
  + "X" — клетка, занятая крестиком.
  + "O" — клетка, занятая ноликом.

**Пример состояния поля:**

[

["X", "", "O"],

["", "X", ""],

["O", "", ""]

]

**1.2. Игроки**

* Игроки представлены строковыми значениями:
  + player1 = "X"
  + player2 = "O"

**1.3. Текущий игрок (self.current\_player)**

* Переменная, указывающая, чей сейчас ход.
* Начальное значение устанавливается функцией выбора первого игрока.
* После каждого хода значение меняется.

**2. Алгоритмы**

**2.1. Алгоритм хода игрока (make\_move)**

Реализует механику выполнения хода:

1. Проверяет, является ли выбранная клетка пустой.
2. Устанавливает символ текущего игрока ("X" или "O") в выбранную клетку.
3. Проверяет, завершена ли игра (победа или ничья).
4. Если игра продолжается, меняет текущего игрока.

**2.2. Алгоритм ИИ (minimax)**

Обеспечивает выполнение хода компьютером:

1. **Оценка текущей позиции:**
   * Возвращает значения для победы компьютера, поражения или ничьей.
2. **Рекурсивный перебор:**
   * Просматривает все возможные ходы для текущего состояния.
3. **Выбор оптимального хода:**
   * Возвращает ход с максимальной выгодой для компьютера.

**2.3. Алгоритм проверки победы (check\_win)**

Проверяет наличие победителя:

* **Строки:** Проверяет, равны ли все элементы каждой строки.
* **Столбцы:** Проверяет равенство элементов в каждом столбце.
* **Диагонали:** Проверяет основные и обратные диагонали.

**2.4. Алгоритм проверки ничьей (check\_draw)**

Проверяет, заполнены ли все клетки игрового поля:

* Если поле заполнено, но победитель не выявлен, фиксируется ничья.

**3. Дополнительные элементы модели**

**3.1. Графический интерфейс (на базе tkinter)**

* **Игровое поле:**
  + Представлено сеткой кнопок (3x3).
  + Кнопки обрабатывают события клика, соответствующие выбору клетки.
* **Информационная панель:**
  + Отображает текущего игрока, победителя или информацию о ничьей.
  + Кнопка для перезапуска игры.

**3.2. Взаимодействие с пользователем**

* Программа отображает текущий ход.
* После завершения игры (победы или ничьей) выводит сообщение.
* Кнопка для выбора режима игры и начала новой партии.

**4. Итеративность проектирования**

Проектирование программы предусматривает поэтапное развитие:

**4.1. Начальные этапы разработки:**

1. Реализация базовых алгоритмов (ход игрока, проверка победы и ничьей).
2. Подключение графического интерфейса для отображения игрового поля.

**4.2. Тестирование и доработка:**

1. Исправление логики и пользовательского интерфейса на основе обратной связи.
2. Тестирование взаимодействия графического интерфейса с игровой логикой.

**4.3. Дополнительные функции (при необходимости):**

* Подсказка для игрока (например, указание оптимального хода).
* Возможность отмены последнего хода.
* Анимация нажатия кнопок для улучшения пользовательского опыта.

**5. Пример взаимодействия компонентов**

1. Пользователь нажимает на кнопку игрового поля.
2. Вызов функции make\_move:
   * Проверяет, можно ли сделать ход.
   * Обновляет self.board и интерфейс кнопок.
3. После хода вызывается:
   * check\_win — проверяет наличие победителя.
   * check\_draw — проверяет, заполнено ли поле.
4. Если игра продолжается:
   * Устанавливается следующий игрок (или вызывается ai\_move, если режим против компьютера).

Модель приложения гибкая, и её структура позволяет добавлять новые функции или изменять существующие механики без значительных изменений в коде.