1. **Структуры данных**

Основные структуры данных, используемые в игре "Крестики-нолики", включают:

* **Игровое поле**: Игровое поле представлено в виде двумерного списка (матрицы) размером 3x3, где каждая клетка может быть пустой или содержать символ 'X' (крестик) или 'O' (нолик).

Пример:

game\_board = [

[None, None, O],

[None, X, None],

[None, None, None]

]

Где:

* None — пустая клетка,
* 'X' — клетка, занятая крестиком,
* 'O' — клетка, занятая ноликом.
* **Игроки**: Игроки представлены в виде строковых значений, обозначающих, какой символ каждый игрок использует.

player1 = 'X'

player2 = 'O'

* **Очередь хода**: Переменная, отвечающая за текущего игрока, будет меняться после каждого хода.

current\_player = player1

1. **Алгоритмы**

Для реализации механики игры потребуются следующие ключевые алгоритмы:

**2.1. Алгоритм хода игрока**

Каждый игрок может поставить свой символ ('X' или 'O') в одну из пустых клеток игрового поля.

Пример:

def make\_move(board, row, col, player):

if board[row][col] is None: # Если клетка пуста

board[row][col] = player # Установить символ игрока

return True

return False

**2.2. Алгоритм проверки победы**

После каждого хода необходимо проверить, завершена ли игра и есть ли победитель. Победа достигается, если три символа одного игрока стоят подряд по горизонтали, вертикали или диагонали.

Пример:

def check\_winner(board):

# Проверка строк

for row in board:

if row[0] == row[1] == row[2] and row[0] is not None:

return row[0]

# Проверка колонок

for col in range(3):

if board[0][col] == board[1][col] == board[2][col] and board[0][col] is not None:

return board[0][col]

# Проверка диагоналей

if board[0][0] == board[1][1] == board[2][2] and board[0][0] is not None:

return board[0][0]

if board[0][2] == board[1][1] == board[2][0] and board[0][2] is not None:

return board[0][2]

return None # Победителя нет

**2.3. Алгоритм проверки ничьей**

Если после всех ходов на поле нет победителя и все клетки заполнены, игра завершается ничьей.

Пример:

def check\_draw(board):

for row in board:

if None in row:

return False # Если есть пустая клетка, это не ничья

return True # Если пустых клеток нет, это ничья

**2.4. Алгоритм смены игрока**

После каждого хода игрок должен смениться на другого.

Пример:

def switch\_player(current\_player, player1, player2):

return player1 if current\_player == player2 else player2

**2.5. Алгоритм перезапуска игры**

Если игра завершилась победой или ничьей, можно перезапустить её, обнулив игровое поле.

Пример:

def reset\_board():

return [

[None, None, None],

[None, None, None],

[None, None, None]

]

1. **Дополнительные элементы модели**

* **Графический интерфейс**: Для отображения игры может быть использован простой графический интерфейс (например, с помощью библиотеки tkinter). Игровое поле будет отображаться в виде кнопок, на которые игроки могут нажимать, чтобы сделать ход.
* **Отображение статуса игры**: На экране будет отображаться информация о текущем игроке, победителе или ничьей.

1. **Итеративность проектирования**

Проектирование игры может включать несколько этапов тестирования и доработок. На каждом этапе могут быть добавлены новые функции (например, возможность отменить ход, подсказки для игрока), а также оптимизированы существующие алгоритмы.

Модель будет дополняться по мере выявления потребностей в ходе тестирования и разработки новых игровых механик.