Урок 2

Крестики-нолики. Создание логики

Базовая логика.

Базовая логика

Инициация игры

Вывод поля на экран

Ход пользователя

Проверка корректности хода

Проверка завершения игры

Базовая логика

Напишем основную логику, которая будет выполнять все действия, кроме хода компьютера. Это отдельная часть программы, которая требует особого внимания.

Создав новый проект в **IDE**, добавим главный класс игры — **TickTackToe**. В нем будет метод **main**. Еще создадим переменные, присущие этому классу.

- **char[][] map** — матрица игры;
- **int SIZE** — размерность поля;
- **символы игры** — крестик, нолик и пустая клетка;
- **экземпляр класса Scanner** для ввода.

Логика в данной версии игры будет делиться на методы, вызываемые из **main**:

1. **initMap()** — инициализация состояния игры.
2. **printMap()** — вывод поля на экран.
3. **Псевдобесконечный цикл while(true)** — для хода игры:
    a. **humanTurn()**;
    b. **computerTurn()**.
4. После каждой фазы проверяем завершение игры — **isEndGame()**. Это проверка любого условия выхода из игры.

Инициация игры

Размещаем в поле игры массив **SIZE x SIZE**, после чего двойным циклом заполняем его пустыми символами.

```
private static void initMap() {
        map = new char[SIZE][SIZE];
        for(int i = 0; i < SIZE; i ++){
            for(int j = 0; j < SIZE; j++){
                map[i][j] = DOT_EMPTY;
            }
        }
    }
```

Вывод поля на экран

При показе поля на экране следует указывать номера строк и столбцов — это удобно. Поэтому понадобятся несколько циклов: один — для вывода «шапки», второй (двойной) — для поля. При этом в каждом шаге внешнего цикла вывода поля первым будет печататься номер строки.

```
private static void printMap() {
        for(int i = 0; i <= SIZE; i++){
            System.out.print(i + " ");
        }
        System.out.println();

        for(int i =0; i < SIZE; i++){
            System.out.print((i+1) + " ");
            for(int j = 0; j < SIZE; j++){
                System.out.print(map[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println();
    }
```

Программа уже умеет начинать игру и печатать поле.

Ход пользователя

Пользователь при ходе будет указывать две его координаты.

Человек не всегда ходит правильно, поэтому надо предусмотреть на 100% корректный ход. Для проверки будем использовать цикл **do..while** — он гарантирует, что хотя бы одна проверка пройдет, а значит ход состоится. Пока координаты не станут корректными, цикл будет повторно запрашивать их у игрока. В проверке будет участвовать отдельный метод **isCellValid(x, y)**.

Игроку удобнее вводить номер строки или столбца при счете с единицы, тогда как компьютер начинает считать с 0. Учтем это при работе с пользовательским вводом и **scanner**.

```
private static void humanTurn() {
        int x, y;
        do {
            System.out.println("Введите координаты ячейки (X Y)");
            y = scanner.nextInt() - 1; // Считывание номера строки
            x = scanner.nextInt() - 1; // Считывание номера столбца
        }
        while(!isCellValid(x, y));

        map[y][x] = DOT_X;
    }
```

Когда получены корректные координаты, цикл завершается. Поэтому сразу после выхода можно обновить состояние игры, проставив в соответствующую ячейку массива символ игрока.

Проверка корректности хода

Метод проверки будет иметь булевский тип, что позволит удобно использовать его в условиях **if**.

Ход будет некорректен, когда он делается вне поля или в занятую ячейку. Это легко задать в условиях простыми сравнениями. В случае проверки выхода за границы поля используем комбинацию условий через логическое **ИЛИ**. Если хотя бы одно условие выполнится, вся конструкция вернет **true**, а результат превратится в **false**.

```
public static boolean isCellValid(int x, int y){
        boolean result = true;

        if(x < 0 || x >= SIZE || y < 0 || y >= SIZE) {
            result = false;
        }

        if(map[y][x] != DOT_EMPTY){
            result = false;
        }

        return result;
}
```

Проверка завершения игры

Остается добавить метод проверки завершения. Для этого выведем игроку поле, чтобы было понятно, что он сделал ход, куда хотел. Затем проверим условие победы и ничьей. Если один из этих методов вернет истину, завершим игру.

Чтобы проверить заполненность поля, проходим по нему, пока не найдем первый пустой символ. Если хотя бы одна ячейка пустует, поле еще не заполнено.

```
public static boolean isCellValid(int x, int y){
        boolean result = true;

        if(x < 0 || x >= SIZE || y < 0 || y >= SIZE) {
            result = false;
        }

        if(map[y][x] != DOT_EMPTY){
            result = false;
        }

        return result;
}
```

Проверка победы реализуется «в лоб» — перечислением восьми комбинаций. Это три одинаковых символа в одном из 8 возможных выигрышных вариантов.

```
private static boolean checkWin(char playerSymbol) {
        boolean result = false;

        if(
            (map[0][0] == playerSymbol && map[0][1] == playerSymbol && map[0][2]
== playerSymbol) ||
        (map[1][0] == playerSymbol && map[1][1] == playerSymbol && map[1][2] ==
playerSymbol) ||
        (map[2][0] == playerSymbol && map[2][1] == playerSymbol && map[2][2] ==
playerSymbol) ||
        (map[0][0] == playerSymbol && map[1][0] == playerSymbol && map[2][0] ==
playerSymbol) ||
        (map[0][1] == playerSymbol && map[1][1] == playerSymbol && map[2][1] ==
playerSymbol) ||
        (map[0][2] == playerSymbol && map[1][2] == playerSymbol && map[2][2] ==
playerSymbol) ||
        (map[0][0] == playerSymbol && map[1][1] == playerSymbol && map[2][2] ==
playerSymbol) ||
        (map[2][0] == playerSymbol && map[1][1] == playerSymbol && map[0][2] ==
playerSymbol)){
                    result = true;
            }

        return result;
    }
```