1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. Кафедра «Высшая школа кибербезопасности»

**КУРСОВАЯ РАБОТА**

1. «Игра «Крестики-нолики» на бесконечном поле»
2. по дисциплине «Структуры данных»
3. Выполнили
4. студенты гр. 5131001/40004 Ишкова Д. А.

Еникеев А.Р.

<*подпись*>

1. Преподаватель
2. ст. преподаватель Вагисаров В.Б.

<*подпись*>

1. «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

Санкт-Петербург

1. 2025

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc200943292)

[ВВЕДЕНИЕ 3](#_Toc200943293)

[1. ОБЗОР ЗАДАЧ И ПРАВИЛ ИГРЫ 4](#_Toc200943294)

[2. АРХИТЕКТУРА ПРОГРАММНОГО РЕШЕНИЯ 6](#_Toc200943295)

[3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА И ИГРОВЫХ МЕХАНИК 9](#_Toc200943296)

[4. ЗАКЛЮЧЕНИЕ 17](#_Toc200943297)

[5. ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА 18](#_Toc200943298)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ 19](#_Toc200943299)

ВВЕДЕНИЕ

В современном мире компьютерные игры остаются одной из самых популярных форм развлечения, а также инструментом для развития логического мышления и стратегического планирования. Одной из самых известных и простых, но в то же время увлекательных игр являются "Крестики-нолики". Классическая версия этой игры ограничена полем 3×3, однако вариация с бесконечным полем открывает новые возможности для тактики и стратегии. Данная курсовая работа посвящена разработке игры "Крестики-нолики на бесконечном поле" с использованием современных технологий программирования.

**Цель работы**

Разработать приложение игры "Крестики-нолики" на языке C с использованием библиотек OpenGL для графики и GLFW для создания окон и обработки ввода. В игре должны быть реализованы бесконечное поле, режим игры для двух игроков с одного компьютера, режим игры пользователя с ботом среднего уровня сложности, обозначение победы или поражения и интерактивное меню.

**Актуальность темы**

Актуальность темы обусловлена тем, что создание игр позволяет развивать навыки программирования, понимание алгоритмов и работу с графическими библиотеками. Кроме того, "Крестики-нолики" — это отличный проект для изучения основ игровой разработки.

**Задачи работы**

1. Изучить основы работы с библиотеками OpenGL и GLFW.
2. Разработать структуру данных для представления бесконечного игрового поля.
3. Реализовать игровую логику, включая управление ходами, алгоритм для ходов бота и проверку победы.
4. Создать графический интерфейс с меню и информационными экранами.
5. Оптимизировать код для обеспечения плавного игрового процесса.
6. ОБЗОР ЗАДАЧ И ПРАВИЛ ИГРЫ

Классическая игра "Крестики-нолики" знакома практически каждому, однако её вариация на бесконечном поле добавляет новые стратегические возможности. В данной реализации игроки могут сражаться друг против друга или против компьютера, а поле не ограничено стандартной сеткой 3×3, что усложняет процесс победы и требует более глубокого анализа.

**1.1 Правила игры**

Игра ведётся на бесконечном поле, разделённом на клетки. Основные правила:

Участники: два игрока (человек против человека или человек против бота).

Игроки по очереди ставят крестики (X) и нолики (O) в свободные клетки.

Цель игры заключается в том, чтобы первым создать непрерывную линию из пяти своих символов по горизонтали, вертикали или диагонали.

Установка символа во время хода осуществляется кликом мыши по нужной клетке поля – программа сама определит очередь и нужный символ.

Программа поддерживает несколько режимов игры: два игрока на одномкомпьютере (поочерёдные ходы); игра против бота (алгоритм средней сложности, основанный на оценке позиции и блокировке игрока).

Игра заканчивается при следующих обстоятельствах**:** победа одного из игроков (5 в ряд) либо ничья (редкий случай на бесконечном поле, но возможен при взаимной блокировке).

**1.2 Постановка задачи**

Разработка игры "Крестики-нолики на бесконечном поле" включает в себя решение следующих задач:

* 1. **Программная реализация на языке C**: игра должна быть написана на языке C с использованием стандартных библиотек.
  2. **Использование графических библиотек**: для отрисовки графики используется OpenGL, а для создания окна и обработки ввода — GLFW.
  3. **Механика проверки победы**: эффективный алгоритм поиска линий из 5 символов (чтобы не перегружать систему на большом поле) и оптимизация проверки, чтобы игра оставалась отзывчивой даже при большом количестве ходов.
  4. **Пользовательский интерфейс**: игра должна содержать стартовое меню с выбором режима игры, выбором первого хода и экран победы.

2. АРХИТЕКТУРА ПРОГРАММНОГО РЕШЕНИЯ

Далее будут рассмотрены некоторые особенности примененные при реализации игры:

**2.1 Структуры данных для представления игровых объектов**

Для представления объектов в игре используются структуры:

1. **Сегмент ходов:** хранит координаты (x, y) каждого сегмента.

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1 – Структура сегмента ходов

1. **Перечисление**: определяет возможное состояние клетки и режим игры.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2 – Структура перечисления

1. **Вспомогательные массивы для графики:** Хранение и рендеринг текста через библиотеку stb\_truetype.

**Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Содержимое, созданное искусственным интеллектом, может быть неверным.**

Рисунок 3 – Массивы для графики

1. **Динамические данные для ИИ:** Хранит возможные ходы для бота (окрестности занятых клеток).

**Изображение выглядит как Шрифт, текст, снимок экрана, Графика

Содержимое, созданное искусственным интеллектом, может быть неверным.**

Рисунок 4 – Данные для ИИ

**2.2 Алгоритмы обработки игрового процесса**

Основные алгоритмы:

* **Основной игровой цикл**: Инициализирует **GLFW** и создаёт окно. Загружает шрифт (Roboto-Regular.ttf). Переходит в режим выбора игры (MENU\_MAIN → MENU\_PVE\_SELECT → GAME\_RUNNING). При закрытии окна освобождает ресурсы.
* **Обработка ввода**: Получает координаты курсора (glfwGetCursorPos). Переводит их в клетку поля. Если клетка свободна (board[cell\_x][cell\_y] == EMPTY) записывает X или O в зависимости от очереди (turn), проверяет победу (check\_win), если игра с ИИ (PLAYER\_VS\_AI), передаёт ход боту (ai\_move).
* **Проверка победы**: проверяет, есть ли 5 одинаковых символов подряд по горизонтали (цикл по строкам), вертикали (цикл по столбцам), диагонали слева направо (двойной цикл + проверка границ), диагонали справа налево (аналогично).
* **Логика ИИ:** Если поле пустое, бот ставит O в центр. Проверяет, может ли выиграть за один ход, блокирует игрока, если тот может выиграть. Оценивает позицию (evaluate\_position), считает "вес" линий для O и штрафует за линии X, выбирает лучший ход (find\_best\_move), перебирает клетки вокруг занятых (get\_candidate\_moves), выбирает ход с максимальной оценкой, если ничего не найдено, делает случайный ход.

**2.3 Основы работы с OpenGL и GLFW**

**OpenGL (Open Graphics Library)** — это кроссплатформенный API для разработки 2D и 3D графики [1]. Он предоставляет функции для отрисовки примитивов (точек, линий, полигонов) и работы с текстурами [2].

**GLFW** — библиотека для создания окон, обработки ввода (клавиатура, мышь) и управления контекстом OpenGL [3].

Основные этапы работы с OpenGL и GLFW [4]:

* 1. Инициализация GLFW и создание окна.
  2. Настройка контекста OpenGL.
  3. Задание проекции (в нашем случае — ортографическая проекция для 2D).
  4. Основной цикл: обработка ввода, обновление состояния, отрисовка.

**2.4 Система координат и примитивы в OpenGL**

В OpenGL используется декартова система координат [2]. Для 2D-графики удобно использовать ортографическую проекцию, которая задаётся следующим образом:

Изображение выглядит как Шрифт, Графика, текст, типография

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 5 – Реализация ортографической проекции

Начало координат (0, 0) — левый верхний **угол** окна. Ось X направлена вправо, ось Y — вниз (как в 2D-графике для экранов). Глубина (near, far) не используется, так как игра 2D.

Примитивы в OpenGL задаются между вызовами glBegin() и glEnd().

Например, для отрисовки сетка поля  используется:

Изображение выглядит как Шрифт, текст, снимок экрана, Графика

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 6 – Примитивы в OpenGL

**2.5 Текстуры и рендеринг текста**

Хотя в данной реализации текстуры не используются, OpenGL поддерживает наложение текстур. Для вывода текста используется вспомогательная библиотека stb\_truetype.

Отрисовка текста:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 8 – Реализация текста

Создание текстуры:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 8 – Реализация текстур

3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА И ИГРОВЫХ МЕХАНИК

**3.1 Функционал разработанного приложения**

Приложение имеет следующие состояния:

* Стартовое меню: выбор режима и выбор первого хода (клавиши 1, 2).

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 9 – Меню программы

* Игровой процесс, начинается сразу при выборе PVP, при выборе PVE нужно еще выбрать кто будет ходить первым.

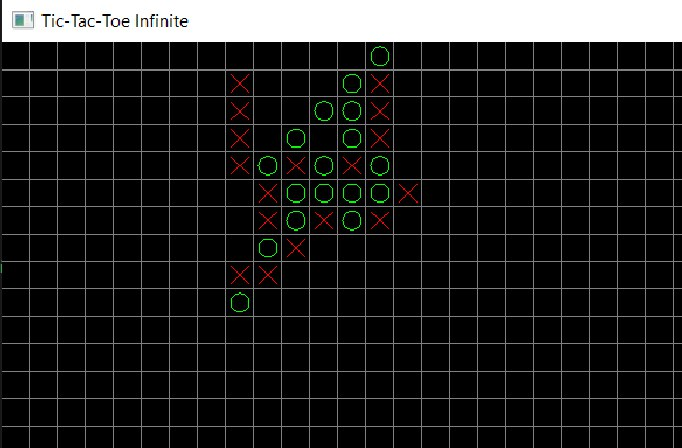


Рисунок 8 – Игровой процесс PVP

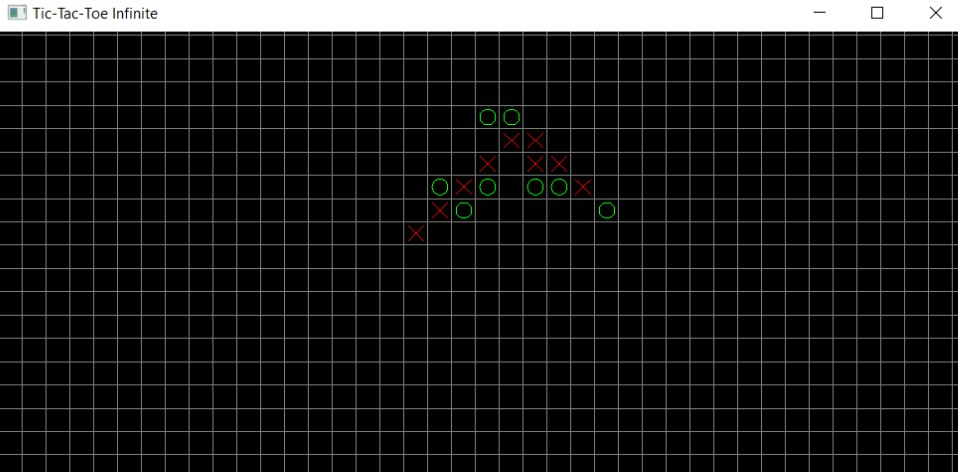


Рисунок 9 – Игровой процесс PVE

**3.2 Реализация графического изображения**

**Меню.** Для меню используется консольный ввод.

Изображение выглядит как текст, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 10 – Реализация меню

**Игровой экран.** Отрисовка фона создаётся автоматически при очистке буфера цвета в основном игровом цикле. Сетка поля отрисовывается функцией draw\_grid(). Крестики (X) - рисуются красными линиями в функции draw\_symbol(), нолики (O) - рисуются зелёными окружностями.

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 11 – Реализация отрисовки игрового поля

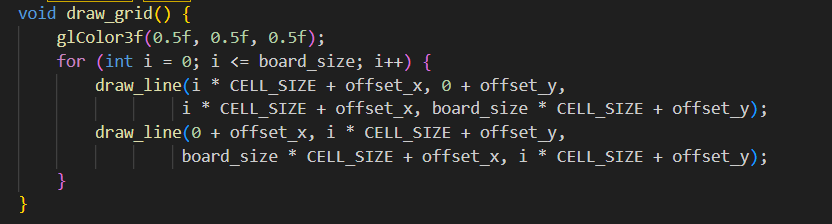


Рисунок 12 – Реализация отрисовки игровой сетки

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 13 – Реализация отрисовки крестиков и ноликов

**Экран завершения игры.** Реализован в консольном вводе при победе одного из участников

**3.3 Система управления в игре**

1. Управление в игре реализовано с использованием мыши для взаимодействия с игровым полем и клавиатуры, обеспечивает интуитивно понятное взаимодействие с игровым процессом на всех этапах. Система обработки ввода построена на функциях библиотеки GLFW.

**Архитектура системы управления:**

1. **Цикл опроса состояния клавиш** - в меню
2. **Обработка мыши** (mouse\_button\_callback) - централизованный обработчик ввода
3. **Обработка клавиатуры** (key\_callback) - централизованный обработчик ввода

**Детализация управления по экранам:**

* 1. **Главное меню:**
     + Клавиши 1 - Режим "Игрок vs Игрок"
     + Клавиши 2 - Режим "Игрок vs ИИ"
     + Клавиши ESC – Выход из игры
  2. **Игровой процесс:**
     + ЛКМ - Постановка символа в выбранную клетку
     + R - Рестарт игры
     + ESC - Выход из игры
  3. **Экран завершения игры:**
     + Клавиша R – рестарт
     + Клавиша ESC – немедленный выход из игры

**3.4 Реализация геймплея и игровой логики**

В нашей реализации используется гибридная модель:

* **Событийная часть**: обработка ходов игрока по клику мыши
* **Автоматическая часть**: мгновенный ответ ИИ при режиме PvE

Полный цикл обработки хода:

* Игрок кликает на свободную клетку
* Система записывает символ (X/O) в board[cell\_x][cell\_y]
* Немедленная проверка победы через check\_win()
* Переключение хода (turn = 1 - turn)
* Если режим PvE - вызов ai\_move()

Система ИИ

Алгоритм работы бота:

* Проверка немедленной победы:
* Блокировка игрока
* Оценочная функция

**3.5 Использованные библиотеки**

Для отображения текста была использована сторонняя библиотека "stb\_truetype.h".

Так же были использована библиотека OpenGl #include <glfw3.h>.

А так же некоторые библиотеки: #include <stdio.h> для работы с текстом внутри кода (sprintf), #include <stdlib.h> - для работы со случайными значениями и другие.

**3.6 Тестирование алгоритма работы бота на играх с другим ИИ**

Для того, чтобы убедиться в эффективности алгоритма ходов бота, было принято решение протестировать его функционал на доступных в интернете уже готовых играх «Крестики-нолики на бесконечном поле» или «Пять в ряд». В ходе проведения нескольких испытаний был сделан вывод: алгоритм бота в созданной программе походит по сложности на алгоритмы других ботов из интернета. Результаты некоторых побед бота разработанной игры представлены ниже.

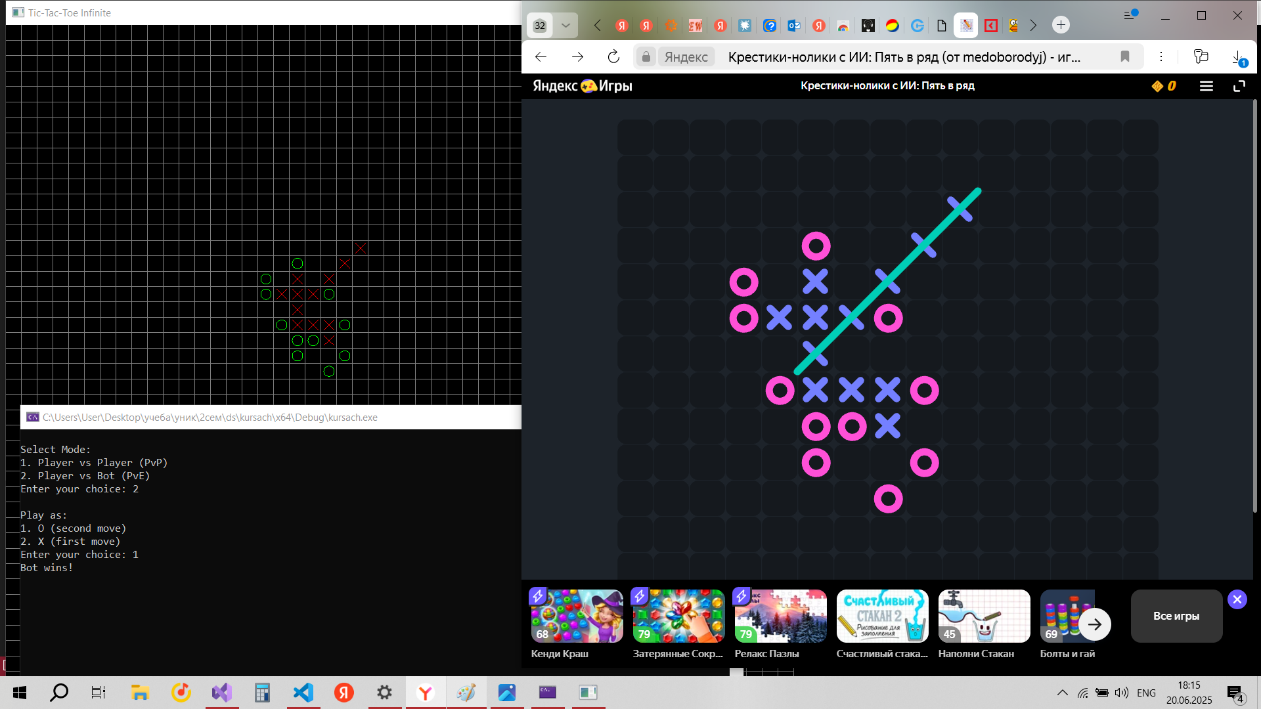


Рисунок 14 — победа №1 бота

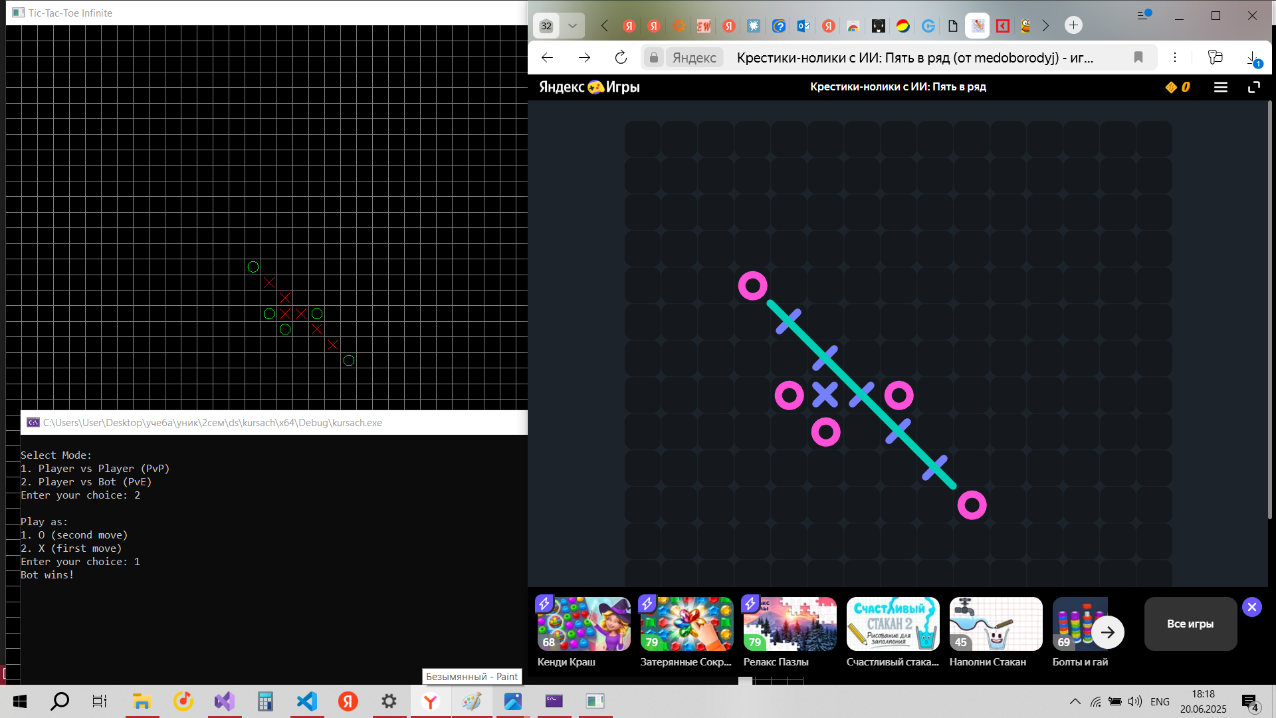


Рисунок 15 — победа №2 бота

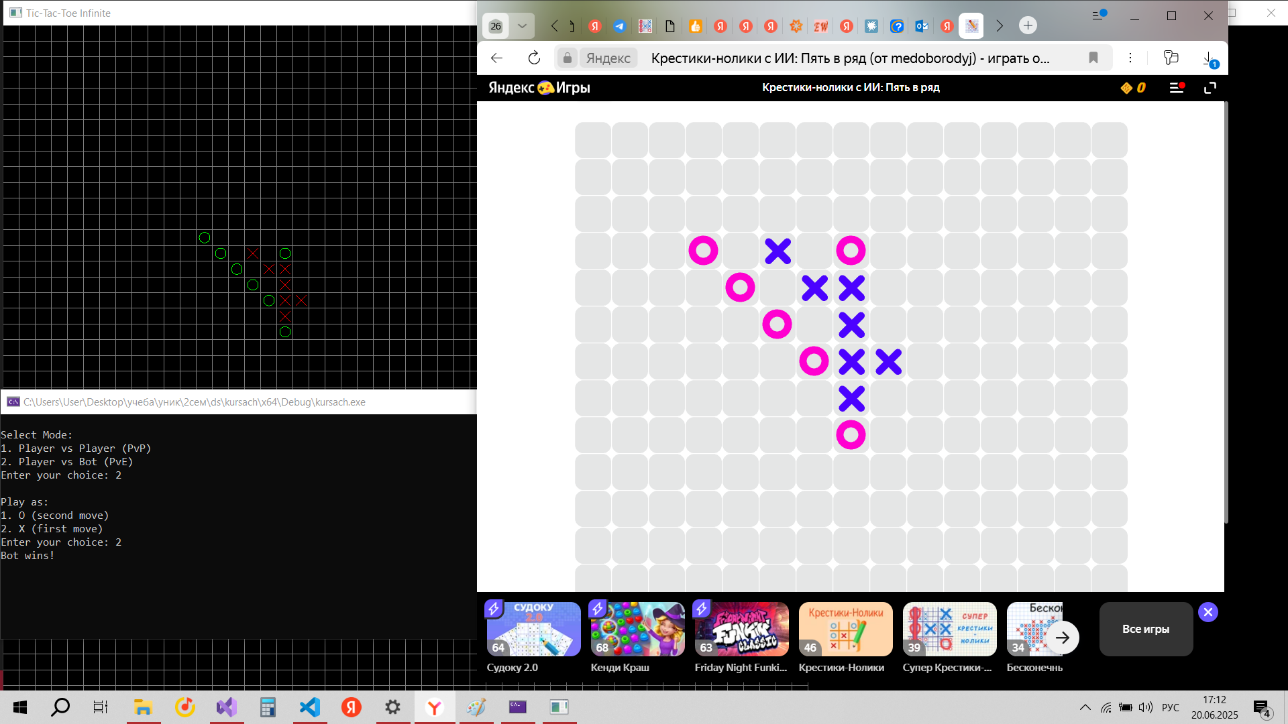


Рисунок 16 — победа №3 бота(в игре в интернете не поставлен последний ход бота, но он есть в окне программы)

4. ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана игра "Крестики-нолики на бесконечном поле" с использованием графической библиотеки OpenGL и библиотеки GLFW для управления окном и вводом. Реализованы все поставленные задачи:

• Создан интуитивно понятный пользовательский интерфейс с меню выбора режима и информационным экраном.

• Реализованы режимы игры с ботом среднего уровня сложности и игры двух пользователей с одного компьютера.

• Работает система совершения ходов и проверки победы игроков.

• Обеспечена плавность игрового процесса за счёт оптимизации.

Игра полностью соответствует классическим правилам "Крестиков-ноликов" и расширена за счёт добавления бесконечного поля. Проект может быть усовершенствован добавлением звуковых эффектов и сохранением рекордов.

5. ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

1. The OpenGL Graphics System: A Specification (Red Book). – [Электронный ресурс]: Silicon Graphics, Inc. – URL: https://www.khronos.org/registry/OpenGL-Refpages/ (дата обращения: 13.06.2025).
2. Дональд Херн, М. Паулин Бейкер "Компьютерная графика и стандарт OpenGL". - 2 изд. - Москва: Вильямс, 2004. - 688 с.
3. GLFW: Reference Documentation [Электронный ресурс]. – 2024. – URL: https://www.glfw.org/docs/latest/ (дата обращения: 13.06.2025)
4. Райт Ричард С., мл., Липчак Бенджамин "OpenGL. Суперкнига". - 3 изд. - Москва: Вильямс, 2006. - 1004 с.
5. Шрайнер Д., Ву М., Нейдер Дж., Девис Т. "OpenGL. Руководство по программированию.". - 4 изд. - Санкт-Петербург: Питер, 2006. - 624 с.

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

#ifndef M\_PI

#define M\_PI 3.14159265358979323846

#endif

#define \_CRT\_SECURE\_NO\_WARNINGS

#define STB\_TRUETYPE\_IMPLEMENTATION

#include "stb\_truetype.h"

#include <GLFW/glfw3.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <stdbool.h>

#include <limits.h>

// Настройки поля

#define MAX\_CELLS 50

#define CELL\_SIZE 24.0f

#define GRID\_LINES 50

typedef enum { EMPTY, X, O } Cell;

typedef enum {

MENU\_MAIN,

MENU\_PVE\_SELECT,

GAME\_RUNNING

} GameState;

typedef enum {

PLAYER\_VS\_PLAYER,

PLAYER\_VS\_AI

} GameMode;

Cell board[MAX\_CELLS][MAX\_CELLS];

int board\_size = GRID\_LINES;

float offset\_x = 0, offset\_y = 0;

int turn = 0; // 0 - X, 1 - O

GameMode mode = PLAYER\_VS\_AI;

GameState state = MENU\_MAIN;

int player\_symbol = EMPTY;

int game\_over = 0;

unsigned char ttf\_buffer[1 << 20]; // ~1MB

unsigned char temp\_bitmap[512 \* 512];

GLuint font\_texture;

stbtt\_bakedchar cdata[96];

// === Прототипы функций ===

void draw\_text(const char\* text, float x, float y);

void reset\_game();

int check\_win(Cell player);

int count\_line(int x, int y, Cell player);

int evaluate\_position(Cell player);

void ai\_move();

void find\_best\_move(int\* best\_x, int\* best\_y);

bool is\_cell\_occupied(int x, int y);

bool is\_cell(int x, int y, Cell player);

void get\_candidate\_moves(int\* candidate\_x, int\* candidate\_y, int\* count);

void draw\_line(float x1, float y1, float x2, float y2);

void draw\_grid();

void draw\_symbol(int x, int y, Cell cell);

void draw\_board();

void mouse\_button\_callback(GLFWwindow\* window, int button, int action, int mods);

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods);

// === MAIN ===

int main() {

if (!glfwInit()) return -1;

GLFWwindow\* window = glfwCreateWindow(1920, 1080, "Tic-Tac-Toe Infinite", NULL, NULL);

if (!window) {

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetMouseButtonCallback(window, mouse\_button\_callback);

glfwSetKeyCallback(window, key\_callback);

glOrtho(0, 1920, 1080, 0, -1, 1);

FILE\* fp = fopen("assets/Roboto-Regular.ttf", "rb");

if (!fp) {

printf("Не могу открыть шрифт!\n");

printf("Проверьте, существует ли файл: assets/Roboto-Regular.ttf\n");

return -1;

}

fread(ttf\_buffer, 1, 1 << 20, fp);

fclose(fp);

glGenTextures(1, &font\_texture);

glBindTexture(GL\_TEXTURE\_2D, font\_texture);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_ALPHA, 512, 512, 0, GL\_ALPHA, GL\_UNSIGNED\_BYTE, NULL);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

stbtt\_BakeFontBitmap(ttf\_buffer, 0, 32.0f, temp\_bitmap, 512, 512, 32, 96, cdata);

reset\_game();

while (state != GAME\_RUNNING) {

printf("\nSelect Mode:\n");

printf("1. Player vs Player (PvP)\n");

printf("2. Player vs Bot (PvE)\n");

printf("Enter your choice: ");

int choice;

scanf("%d", &choice);

if (choice == 1) {

mode = PLAYER\_VS\_PLAYER;

state = GAME\_RUNNING;

turn = 0;

reset\_game();

} else if (choice == 2) {

printf("\nPlay as:\n");

printf("1. O (second move)\n");

printf("2. X (first move)\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

mode = PLAYER\_VS\_AI;

player\_symbol = O;

turn = 1;

state = GAME\_RUNNING;

reset\_game();

ai\_move(); // Бот делает первый ход как X

turn = 0;

if (check\_win(X)) {

printf("Bot wins!\n");

game\_over = 1;

}

} else if (choice == 2) {

mode = PLAYER\_VS\_AI;

player\_symbol = X;

turn = 0;

state = GAME\_RUNNING;

reset\_game();

}

}

}

while (!glfwWindowShouldClose(window)) {

glClear(GL\_COLOR\_BUFFER\_BIT);

draw\_grid();

draw\_board();

glfwSwapBuffers(window);

glfwPollEvents();

}

glfwDestroyWindow(window);

glfwTerminate();

return 0;

}

// Функция отрисовки текста

void draw\_text(const char\* text, float x, float y) {

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, font\_texture);

glColor3f(1.0f, 1.0f, 1.0f);

glBegin(GL\_QUADS);

float xPos = x;

float yPos = y;

while (\*text) {

if (\*text >= 32 && \*text < 128) {

stbtt\_bakedchar\* b = &cdata[\*text - 32];

float x0 = b->x0 + xPos;

float y0 = b->y0 + yPos;

float x1 = b->x1 + xPos;

float y1 = b->y1 + yPos;

float s0 = b->x0 / 512.0f;

float t0 = b->y0 / 512.0f;

float s1 = b->x1 / 512.0f;

float t1 = b->y1 / 512.0f;

glTexCoord2f(s0, t0); glVertex2f(x0, y0);

glTexCoord2f(s1, t0); glVertex2f(x1, y0);

glTexCoord2f(s1, t1); glVertex2f(x1, y1);

glTexCoord2f(s0, t1); glVertex2f(x0, y1);

xPos += b->xadvance;

}

++text;

}

glEnd();

glDisable(GL\_TEXTURE\_2D);

}

// Сброс игры

void reset\_game() {

for (int i = 0; i < MAX\_CELLS; i++)

for (int j = 0; j < MAX\_CELLS; j++)

board[i][j] = EMPTY;

game\_over = 0;

if (mode == PLAYER\_VS\_AI) {

if (player\_symbol == X) {

turn = 0;

} else {

turn = 1;

}

} else {

turn = 0;

}

}

// Проверка победы при 5 в ряд

int check\_win(Cell player) {

// По горизонтали

for (int y = 0; y < board\_size; y++) {

int count = 0;

for (int x = 0; x < board\_size; x++) {

if (board[x][y] == player)

count++;

else

count = 0;

if (count >= 5) return 1;

}

}

// По вертикали

for (int x = 0; x < board\_size; x++) {

int count = 0;

for (int y = 0; y < board\_size; y++) {

if (board[x][y] == player)

count++;

else

count = 0;

if (count >= 5) return 1;

}

}

// Диагонали слева направо

for (int start\_x = 0; start\_x < board\_size; start\_x++) {

for (int start\_y = 0; start\_y < board\_size; start\_y++) {

int count = 0;

for (int i = 0; i < board\_size; i++) {

int x = start\_x + i;

int y = start\_y + i;

if (x >= board\_size || y >= board\_size) break;

if (board[x][y] == player)

count++;

else

count = 0;

if (count >= 5) return 1;

}

}

}

// Диагонали справа налево

for (int start\_x = 0; start\_x < board\_size; start\_x++) {

for (int start\_y = 0; start\_y < board\_size; start\_y++) {

int count = 0;

for (int i = 0; i < board\_size; i++) {

int x = start\_x + i;

int y = start\_y - i;

if (x >= board\_size || y < 0) break;

if (board[x][y] == player)

count++;

else

count = 0;

if (count >= 5) return 1;

}

}

}

return 0;

}

// Подсчёт веса линии вокруг точки

int count\_line(int x, int y, Cell player) {

const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};

int score = 0;

for (int d = 0; d < 4; d++) {

int dx = dirs[d][0], dy = dirs[d][1];

int length = 1;

int open\_ends = 0;

// В одном направлении

for (int step = 1; step <= 5; step++) {

int nx = x + dx \* step;

int ny = y + dy \* step;

if (nx < 0 || nx >= board\_size || ny < 0 || ny >= board\_size) {

open\_ends++;

break;

}

if (board[nx][ny] == player) length++;

else break;

}

// В другом направлении

for (int step = 1; step <= 5; step++) {

int nx = x - dx \* step;

int ny = y - dy \* step;

if (nx < 0 || nx >= board\_size || ny < 0 || ny >= board\_size) {

open\_ends++;

break;

}

if (board[nx][ny] == player) length++;

else break;

}

if (length >= 5) score += 100000;

else if (length == 4 && open\_ends >= 1) score += 10000;

else if (length == 3 && open\_ends >= 1) score += 1000;

else if (length == 2 && open\_ends >= 1) score += 100;

}

return score;

}

// Оценка позиции для бота

int evaluate\_position(Cell player) {

int score = 0;

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] == player) {

score += count\_line(x, y, player);

} else if (board[x][y] != EMPTY) {

score -= count\_line(x, y, board[x][y]);

}

}

}

return score;

}

// Получить список возможных ходов (окрестности занятых клеток)

void get\_candidate\_moves(int\* candidate\_x, int\* candidate\_y, int\* count) {

\*count = 0;

bool occupied[100][100] = {0};

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] != EMPTY) {

for (int dx = -1; dx <= 1; dx++) {

for (int dy = -1; dy <= 1; dy++) {

int nx = x + dx;

int ny = y + dy;

if (nx >= 0 && nx < board\_size && ny >= 0 && ny < board\_size &&

board[nx][ny] == EMPTY && !occupied[nx][ny]) {

candidate\_x[\*count] = nx;

candidate\_y[\*count] = ny;

(\*count)++;

occupied[nx][ny] = true;

}

}

}

}

}

}

if (\*count == 0) {

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] == EMPTY) {

candidate\_x[\*count] = x;

candidate\_y[\*count] = y;

(\*count)++;

return;

}

}

}

}

}

// Найти лучший ход

void find\_best\_move(int\* best\_x, int\* best\_y) {

int max\_score = -1000000;

\*best\_x = -1;

\*best\_y = -1;

int candidate\_x[1000], candidate\_y[1000];

int candidate\_count = 0;

get\_candidate\_moves(candidate\_x, candidate\_y, &candidate\_count);

for (int i = 0; i < candidate\_count; i++) {

int x = candidate\_x[i];

int y = candidate\_y[i];

if (board[x][y] == EMPTY) {

board[x][y] = O;

int score = evaluate\_position(O);

board[x][y] = EMPTY;

if (score > max\_score) {

max\_score = score;

\*best\_x = x;

\*best\_y = y;

}

}

}

if (\*best\_x == -1) {

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] == EMPTY) {

\*best\_x = x;

\*best\_y = y;

return;

}

}

}

}

}

// Ход бота (минимакс)

void ai\_move() {

if (game\_over) return;

int best\_x = -1, best\_y = -1;

// Если это первый ход — занимаем центр

int total\_moves = 0;

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] != EMPTY) total\_moves++;

}

}

if (total\_moves == 0) {

board[board\_size / 2][board\_size / 2] = O;

return;

}

// Если есть выигрышные ходы — делаем их

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] == EMPTY) {

board[x][y] = O;

if (check\_win(O)) return;

board[x][y] = EMPTY;

}

}

}

// Если нужно блокировать игрока — делаем это

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] == EMPTY) {

board[x][y] = X;

if (check\_win(X)) {

board[x][y] = O;

return;

}

board[x][y] = EMPTY;

}

}

}

// Ищем лучший ход по оценке

find\_best\_move(&best\_x, &best\_y);

if (best\_x != -1 && best\_y != -1) {

board[best\_x][best\_y] = O;

return;

}

// Если не нашли — случайный ход

for (int x = 0; x < board\_size; x++) {

for (int y = 0; y < board\_size; y++) {

if (board[x][y] == EMPTY) {

board[x][y] = O;

return;

}

}

}

}

// Рисование линии

void draw\_line(float x1, float y1, float x2, float y2) {

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

// Рисование сетки

void draw\_grid() {

glColor3f(0.5f, 0.5f, 0.5f);

for (int i = 0; i <= board\_size; i++) {

draw\_line(i \* CELL\_SIZE + offset\_x, 0 + offset\_y,

i \* CELL\_SIZE + offset\_x, board\_size \* CELL\_SIZE + offset\_y);

draw\_line(0 + offset\_x, i \* CELL\_SIZE + offset\_y,

board\_size \* CELL\_SIZE + offset\_x, i \* CELL\_SIZE + offset\_y);

}

}

// Рисование крестика или нолика

void draw\_symbol(int x, int y, Cell cell) {

float cx = x \* CELL\_SIZE + offset\_x + CELL\_SIZE / 2;

float cy = y \* CELL\_SIZE + offset\_y + CELL\_SIZE / 2;

float r = CELL\_SIZE / 3;

if (cell == X) {

glColor3f(1.0f, 0.0f, 0.0f);

draw\_line(cx - r, cy - r, cx + r, cy + r);

draw\_line(cx - r, cy + r, cx + r, cy - r);

} else if (cell == O) {

glColor3f(0.0f, 1.0f, 0.0f);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 360; i++) {

float rad = i \* M\_PI / 180;

float x = cx + cos(rad) \* r;

float y = cy + sin(rad) \* r;

glVertex2f(x, y);

}

glEnd();

}

}

// Отрисовка всего игрового поля

void draw\_board() {

for (int i = 0; i < board\_size; i++) {

for (int j = 0; j < board\_size; j++) {

if (board[i][j] != EMPTY)

draw\_symbol(i, j, board[i][j]);

}

}

}

// Обработка кликов мыши

void mouse\_button\_callback(GLFWwindow\* window, int button, int action, int mods) {

if (game\_over || button != GLFW\_MOUSE\_BUTTON\_1 || action != GLFW\_PRESS)

return;

double x, y;

glfwGetCursorPos(window, &x, &y);

int cell\_x = (x - offset\_x) / CELL\_SIZE;

int cell\_y = (y - offset\_y) / CELL\_SIZE;

if (cell\_x >= 0 && cell\_x < board\_size && cell\_y >= 0 && cell\_y < board\_size &&

board[cell\_x][cell\_y] == EMPTY) {

board[cell\_x][cell\_y] = (turn == 0) ? X : O;

if (check\_win(board[cell\_x][cell\_y])) {

printf("Player %c wins!\n", board[cell\_x][cell\_y] == X ? 'X' : 'O');

game\_over = 1;

}

turn = 1 - turn;

if (mode == PLAYER\_VS\_AI && turn == 1 && !game\_over) {

ai\_move();

turn = 0;

if (check\_win(O)) {

printf("Bot wins!\n");

game\_over = 1;

}

}

}

}

// Обработка клавиш

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods) {

if (action == GLFW\_PRESS && key == GLFW\_KEY\_R) {

reset\_game();

game\_over = 0;

state = MENU\_MAIN;

}

}