

**МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА
РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение высшего
образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.Т. ТРУБИЛИНА»**

Факультет заочного обучения

Кафедра системного анализа и обработки информации

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ**

по дисциплине «Алгоритмизация и программирование»

на тему «Программирование игрового приложения „Усложнённые
крестики-нолики“»

Направление подготовки 09.03.05 «Прикладная информатика»

Направленность «Менеджмент проектов в области информационных
технологий, создание и поддержка информационных систем»

Выполнил:
Сидоров Дмитрий Ильич
группа ПИЗ1902
Руководитель:
к.т.н., доцент
Орлянская Н.П.

Дата защиты _____
Оценка _____
Орлянская Н.П.

Краснодар 2020

РЕФЕРАТ

48 с., 7 рис., 3 библ., 17 прил.

ПРОГРАММИРОВАНИЕ ИГРОВОГО ПРИЛОЖЕНИЯ «УСЛОЖНЁННЫЕ КРЕСТИКИ-НОЛИКИ»

Ключевые слова: паттерны проектирования, теория игр, минимакс, альфа-бета отсечение, кроссплатформенность, графика.

Цель работы: разработка графического приложения «Усложнённые крестики-нолики» на языке высокого уровня C++.

Объект исследования: алгоритм принятия решений.

Предмет исследования: средства языка программирования C++ для реализации 2D-графики и алгоритмизации принятия решений.

Полученные результаты: разработанное приложение позволяет осуществлять игру в «Крестики-нолики» между пользователем и компьютером с учётом возможности настраивания размера игрового поля и передачи права первого хода.

ОГЛАВЛЕНИЕ

Введение.....	4
1 Описание предметной области.....	5
1.1 Постановка задачи.....	5
1.2 Сведения из теории.....	5
1.3 Выбор инструментальных средств.....	7
2 Технология разработки приложения.....	9
2.1 Алгоритм решения.....	9
2.2 Описание интерфейса приложения.....	10
2.3 Описание программы.....	12
2.4 Результаты работы программы.....	14
3 Руководство пользователя.....	16
Заключение.....	17
Список использованных источников.....	18
Приложение 1.....	19
Приложение 2.....	22
Приложение 3.....	23
Приложение 4.....	28
Приложение 5.....	29
Приложение 6.....	34
Приложение 7.....	35
Приложение 8.....	39
Приложение 9.....	40
Приложение 10.....	41
Приложение 11.....	42
Приложение 12.....	43
Приложение 13.....	44
Приложение 14.....	45
Приложение 15.....	46
Приложение 16.....	47
Приложение 17.....	48

ВВЕДЕНИЕ

К настоящему времени разработано большое количество различных реализаций игры «Крестики-нолики». Методы, используемые в данной курсовой работе, основаны на использовании паттернов проектирования: состояния, шаблонного метода, синглтона и других.

Для нахождения оптимального хода компьютера используется простейший из методов теории принятия решений – алгоритм минимакс. Для улучшения производительности алгоритма дополнительно применяется альфа-бета отсечение.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 ПОСТАНОВКА ЗАДАЧИ

Необходимо реализовать графическое приложение для игры в «Крестики-нолики».

Игра должна происходить между пользователем и компьютером и иметь настройки размерности поля и возможность передачи права первого хода компьютеру.

Приложение необходимо реализовать на языке высокого уровня C++ с поддержкой кроссплатформенности – возможностью сборки для персональных компьютеров с операционными системами на основе ядра Linux и семейства Windows.

1.2 СВЕДЕНИЯ ИЗ ТЕОРИИ

Процесс игры можно представить в виде дерева вариантов. В каждой конкретной позиции игрок имеет выбор между разными вариантами следующего хода, учитывая при этом дальнейшие варианты развития игровой партии.

Компьютер при выборе исходит из максимизации своей выгоды и минимизации выгоды игрока – такой алгоритм носит название «минимакс». Функция, которая оценивает игровое поле, называется статической функцией оценки (далее – СФО), чем «выгоднее» игровая ситуация, тем большее значение СФО [2, с. 54].

Опишем минимакс как рекурсивную функцию:

1. Если найдено конечное состояние, возвращается результат СФО.
2. Проходит по всем пустым ячейкам игрового поля и вызывает для каждой рекурсивно минимакс.
3. Получает для каждого вызова значения СФО, оценивает их и возвращает лучшее.

Для демонстрации работы рассмотрим пошагово эндшпиль одной из партий (Рисунок 1.1):

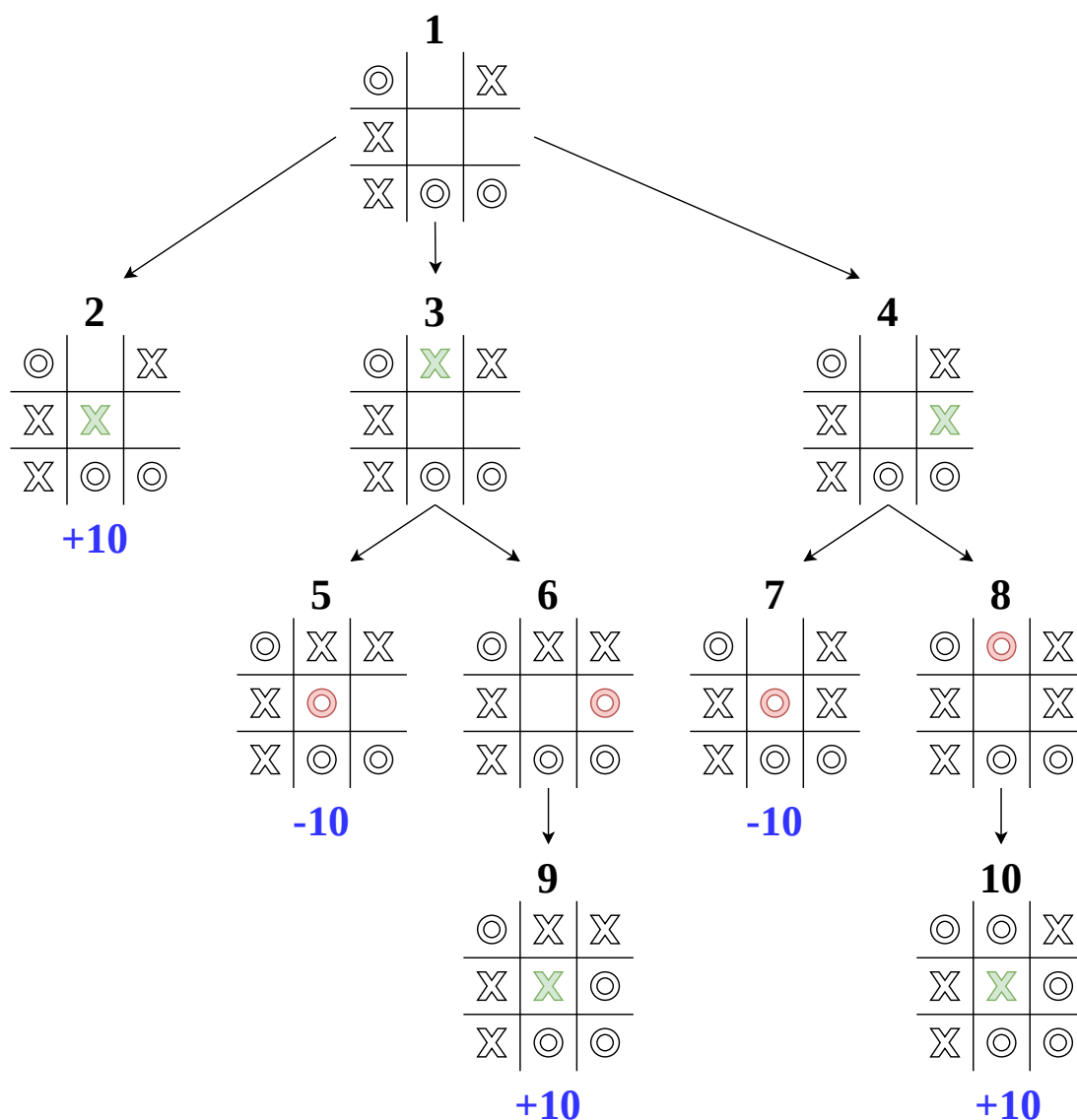


Рисунок 1.1 – Схема вариантов завершения игры

- Состояние 1 для X порождает состояния 2, 3 и 4 и вызывает минимакс на этих состояниях.
- Состояние 2 даёт оценку +10 к состоянию 1, потому что здесь игра заканчивается победой.

- Состояния 3 и 4 не являются конечными состояниями, так что состояние 3 порождает состояния 5 и 6 и вызывает минимакс на них, пока состояние 4 делает то же самое с состояниями 7 и 8.
- Состояние 5 даёт оценку -10 к состоянию 3, также состояние 7 даёт -10 очков к состоянию 4.
- Состояния 6 и 8 порождают по одному возможному конечному состоянию, которые дают оценку +10 к состояниям 3 и 4.
- Так как в порождённых состояниях состояниями 3 и 4 ходят нолики, нолики ищут минимальную оценку и делают выбор между -10 и +10, состояния 3 и 4 получают -10 каждое.
- Наконец состояния 2, 3 и 4 имеют оценки +10, -10, -10 соответственно, и в состоянии 1 алгоритм, максимизируя ход, выбирает состояние 2, дающее оценку +10. Партия завершена.

Количество рекурсивных вызовов минимакса экспоненциально растёт при изменении величины игрового поля. Чтобы компенсировать рост числа операций, применяется дополнительное альфа-бета отсечение. Вводятся коэффициенты альфа (минимальное значение СФО в ветке) и бета (максимальное значение СФО в ветке). На каждом уровне происходит сравнение текущей СФО с коэффициентами альфа и бета, что позволяет отсекаать заведомо менее выгодные для компьютера или более выгодные для игрока [3, с. 541-543].

1.3 ВЫБОР ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ

В качестве основного инструмента разработки был выбран бесплатный текстовый редактор с открытым исходным кодом Atom, разработанный компанией GitHub, Inc.

Редактор является легковесным, поддерживает систему управления версиями Git и предоставляет расширяемость с помощью пакетов, устанавливаемых дополнительно.

В частности были использованы следующие из них:

1. atom-ide-ui – расширение интерфейса редактора для поддержки языковых сервисов и возможностей отладчика.
2. autocomplete-clang – автодополнение для C/C++/Objective-C с помощью API, предоставляемого Clang.
3. atom-ide-debugger-native-gdb – интеграция нативного отладчика GDB с интерфейсом редактора.

Для разработки был выбран стандарт C++17, так как имеет место использование синтаксиса распаковки пар, который отсутствует в более ранних версиях языка [1].

Учитывая игровую специфику приложения, для разработки была использована свободная кроссплатформенная мультимедийная библиотека SFML (Simple and Fast Multimedia Library), позволяющая легко взаимодействовать с 2D графикой.

Для автоматизации сборки используется CMake, непосредственно компилятором под Linux-системами выступает GCC, под Windows – MinGW.

2 ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

2.1 АЛГОРИТМ РЕШЕНИЯ

Для решения поставленной задачи реализуется разделение игры на непосредственно игровое поле и графический интерфейс пользователя.

На Рисунке 2.1 представлена общая схема взаимоотношений классов приложения.

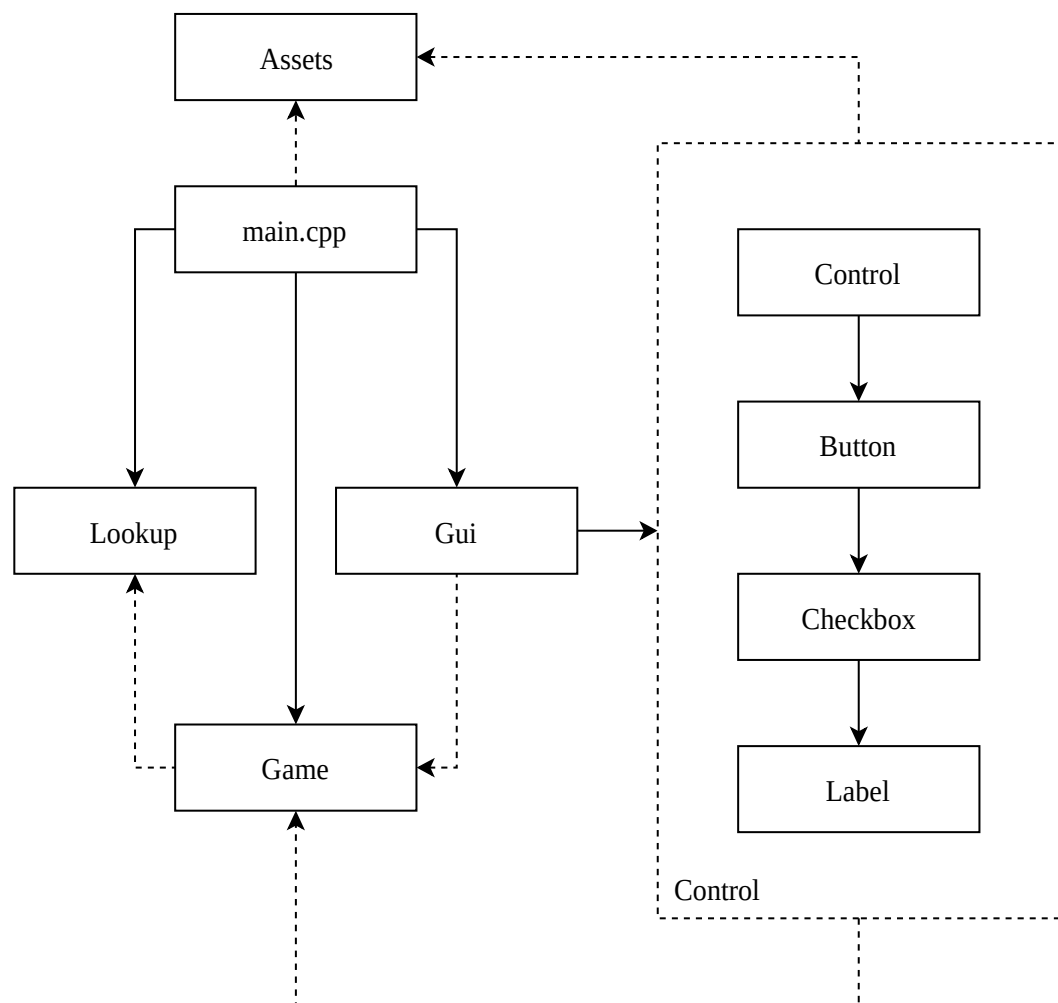


Рисунок 2.1 – Структура приложения

В функции `int main()` инициализируются объекты типов `Lookup`, `Game`, `Gui` (см. Приложение 1). `Lookup` реализует алгоритм нахождения наилучшего хода компьютера (см. Приложения 2 и 3), взаимодействие с алгоритмом осуществляет через объект типа `Game`.

Game реализует игровое поле (см. Приложения 4 и 5), взаимодействие с которым может осуществляться с помощью графического интерфейса пользователя, предоставляемого объектом типа Gui.

При вызове конструктора копирования Gui создаются элементы графического интерфейса пользователя (см. Приложения 6 и 7), представляемые общим родительским классом Control (см. Приложение 8).

Control имеет доступ к классу Assets, в котором хранятся ресурсы приложения (см. Приложения 9 и 10), также имеет доступ к объекту Game в функциях клика по элементам.

В качестве алгоритма Lookip используется минимакс с альфа-бета отсечением. СФО на каждой глубине прохождения дерева вариантов возвращает три состояния: победа, поражение, ничья (или неоконченная игра).

2.2 ОПИСАНИЕ ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ

Окно приложения состоит из двух частей: игрового поля слева и графического интерфейса пользователя справа (см. Рисунок 2.2).

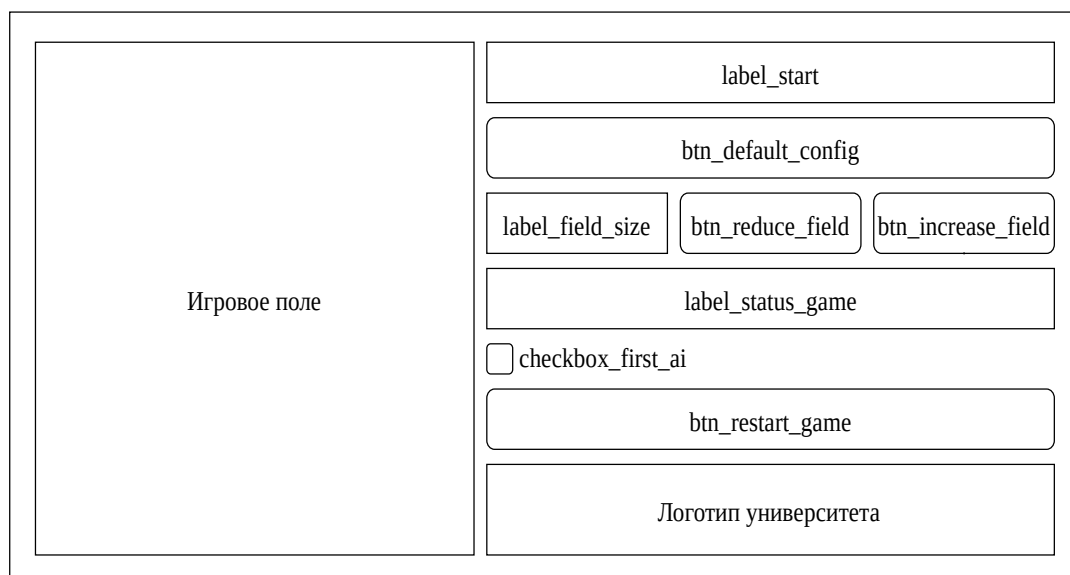


Рисунок 2.2 – Макет приложения

Игровое поле представляет собой матрицу размером $N \times N$ без внешнего обрамления, где каждая ячейка может иметь только одно из трёх состояний: пустое, заполненное крестиком, заполненное ноликом.

Крестики отображаются как пересекающиеся диагонали квадрата и имеют синий цвет, нолики отображаются как окружность и имеют красный цвет. Пустая ячейка не имеет специфических выделений.

В случае выстраивания N фигур одного типа в ряд или диагональ отображается выделение этого ряда или диагонали цветной линией, где цвет соответствует «зачёркиваемому» типу фигур.

Для реализации различных настроек и возможности перезапуска игры реализован графический интерфейс пользователя. Графический интерфейс пользователя состоит из элементов трёх типов: текст (см. Приложения 11 и 12), кнопка (см. Приложения 13 и 14) и чекбокс (см. Приложения 15 и 16). Информационно в этой области присутствует логотип КубГАУ.

Далее следует описание каждого из элементов:

1. `label_start` – приветствующее сообщение с названием игры.
2. `btn_default_config` – кнопка сброса размера игрового поля на 3×3 .
3. `label_field_size` – информация о текущем размере игрового поля.
4. `btn_reduce_field` – кнопка уменьшения размерности игрового поля.
5. `btn_increase_field` – кнопка увеличения размерности игрового поля.
6. `label_status_game` – информация о текущем состоянии игры (игра не закончена, победа крестиков, победа ноликов, ничья).
7. `checkbox_first_ai` – чекбокс, передающий право первого хода от игрока компьютеру.
8. `btn_restart_game` – кнопка перезапуска игры с новыми настройками.

2.3 ОПИСАНИЕ ПРОГРАММЫ

В функции `int main()` осуществляется инициализация `Assets` с последующей загрузкой ресурсов (шрифт, логотип университета), происходит инициализация окна приложения, инициализация объектов игры (`Game`) и графического интерфейса пользователя (`Gui`). Здесь расположен цикл, обрабатывающий входные сигналы пользователя (перемещение курсора, нажатие кнопок, изменение размера окна) и передающий управление отрисовкой состояний в окне соответствующим элементам приложения.

Для хранения ресурсов игры используется класс `Assets`, являющийся по своей сути синглтоном Майерса, его использование обуславливается необходимостью обращений к статичным объектам-ресурсам из разных частей приложения, в результате которых могли бы порождаться множественные копии класса `Assets`.

Объект типа `Game` отвечает за отрисовку игрового поля, обработку нажатий на соответствующие ячейки внутри него. Через объект такого типа происходит взаимодействие с объектом типа `Lookup`, реализующий ходы компьютера.

Максимальная глубина вхождения в дерево вариантов в `Lookup` ограничена константой `kMaxDepth` (по-умолчанию равной 4). Ограничение установлено для получения результата работы алгоритма в адекватное время ввиду присутствия возможности изменения размерности игрового поля: поле размером 3×3 порождает 19683 (3^9) возможных состояний, 4×4 – уже 43046721 (3^{16}), 5×5 – 847288609443 (3^{25}) и так далее. Основная функция `std::pair<int, std::pair<int, int>> Lookup::MinimaxOptimization(`

`std::vector<std::vector<StateCell>>& board, StateCell marker, int depth,`
`int alpha, int beta)`

есть сам алгоритм минимакс. При первом вызове в `board` передаётся текущее состояние игрового поля, в `marker` – идентификатор (крестик `StateCell:X` или

нолик StateCell:O), под которым играет компьютер, глубина depth равна 0, alpha и beta равны kLoss и kWin соответственно.

Объект типа Gui при инициализации вызывает функцию void Gui::Initialize(), в которой происходит инициализация компонентов пользовательского интерфейса приложения (см. Рисунок 1), заполнение контейнера элементов std::map<std::string, std::shared_ptr<Control>> controls_, для дальнейшего доступа к ним. Здесь обрабатываются перемещения курсора по элементам интерфейса, нажатие на них.

Все элементы интерфейса являются наследниками класса Control, имеющий следующие поля: std::wstring title_text_ – название, bool is_hovered_ – состояние нахождения под указателем, bool is_pressed_ – состояния «нажатия», sf::Vector2f size_ – размер, sf::Vector2f position_ – положение внутри контейнера пользовательского интерфейса. Публичное поле void (*OnClick) (Control& me, Game& game, Gui& gui) предназначено для назначения обработчика нажатия на соответствующий элемент интерфейса.

Каждый элемент интерфейса обязательно имеет виртуальную функцию virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const, которая вызывается всякий раз при отрисовке интерфейса и задаёт внешний вид элемента: контуры, цвета, текст.

В Config.h константно задаются стандартные размеры окна, элементов (если их размер не изменён при инициализации в Gui.cpp), пути к ресурсным файлам, размер шрифта, ограничение глубины минимакса, значения СФО в случае победы, ничьи и проигрыша (см. Приложение 17).

2.4 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

На Рисунке 2.3 представлено окно приложения сразу после запуска.

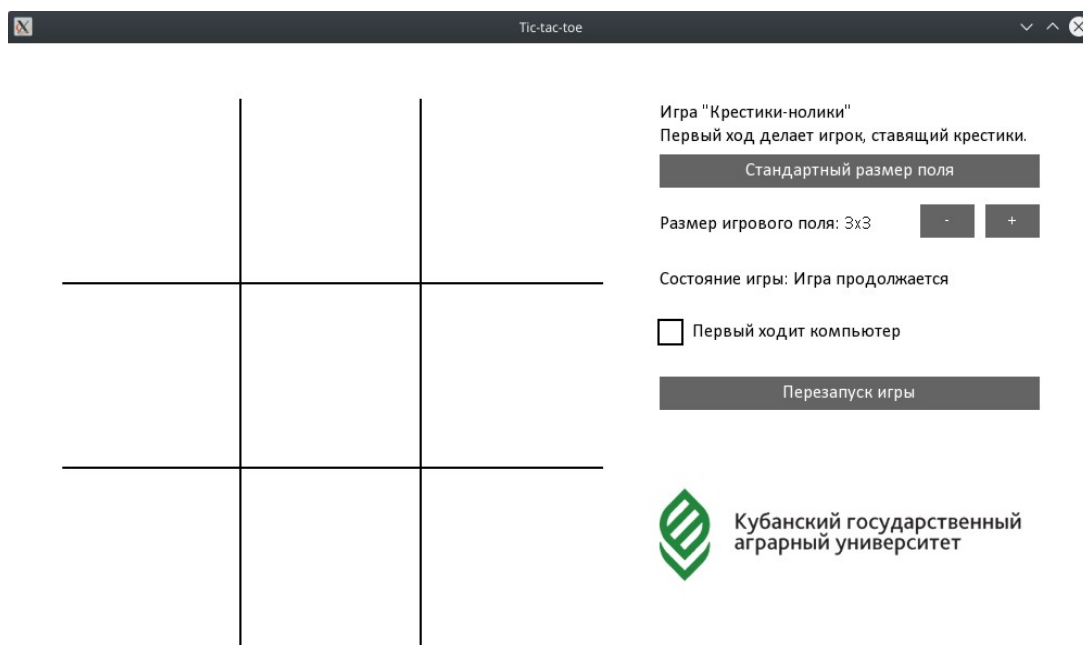


Рисунок 2.3 – Окно приложения после запуска

На Рисунке 2.4 представлена игровая партия с победой ноликов.

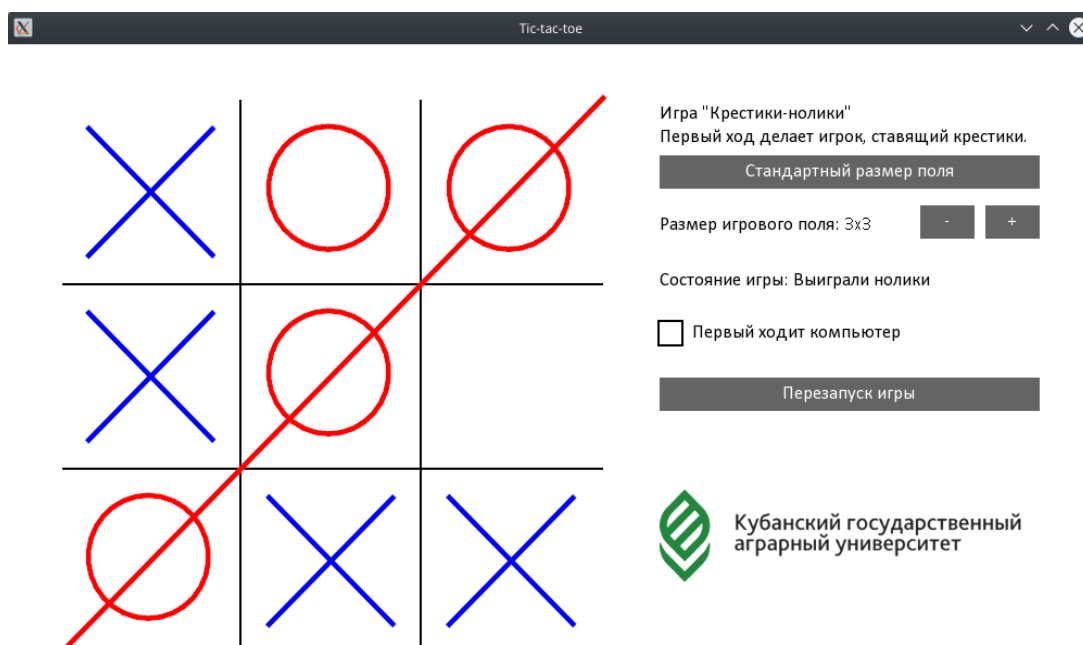


Рисунок 2.4 – Законченная игровая партия

На Рисунке 2.5 представлена законченная игровая партия на поле размером 5x5 с первым ходом компьютера.

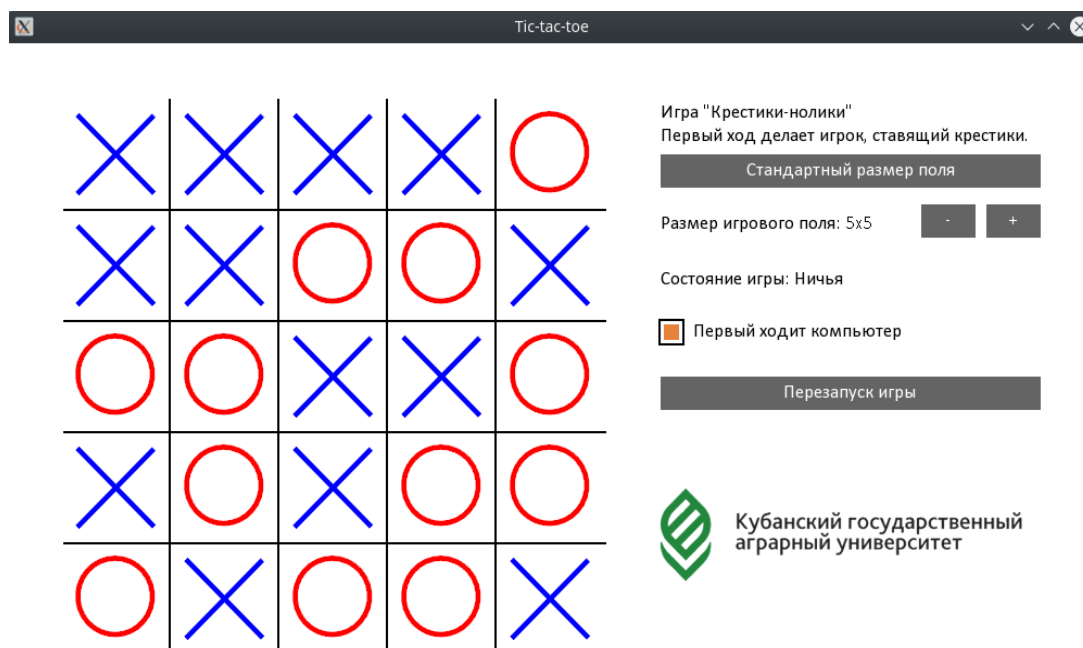


Рисунок 2.5 – Игра с первым ходом компьютера

На Рисунке 2.6 представлено запущенное приложение под операционной системой Windows 10.

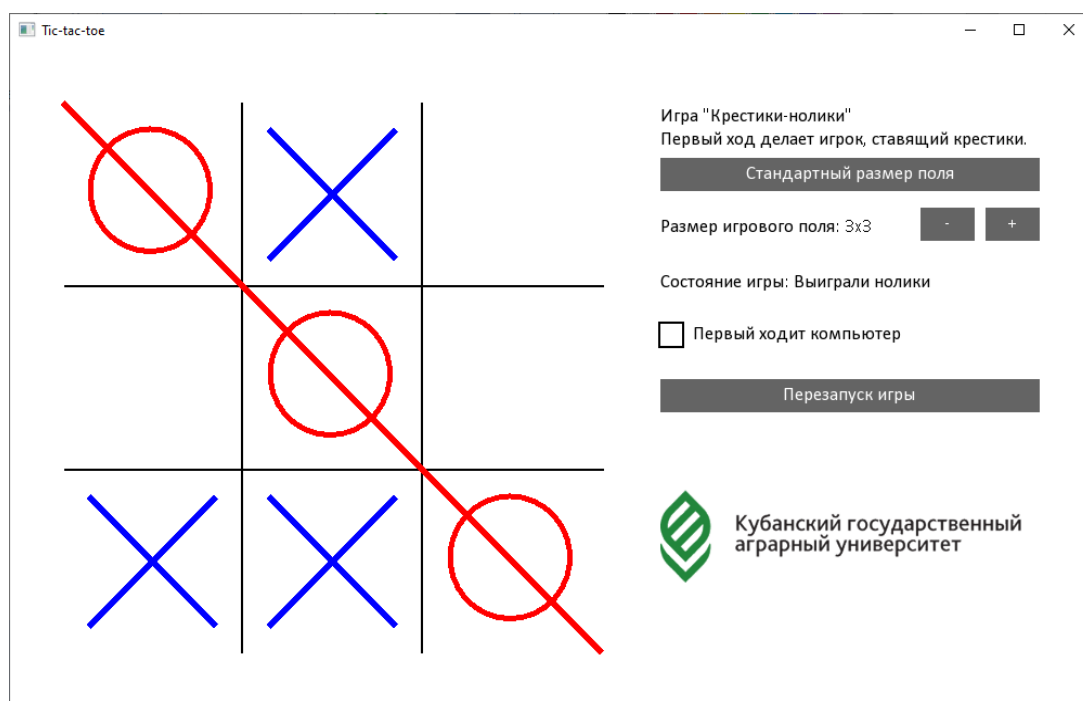


Рисунок 2.6 – Окно приложения в ОС Windows 10

3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Приложение изначально разработано для операционных систем на базе ядра Linux, для работы требуется установленный пакет `libsFML-dev`, доступный во многих репозиториях.

Для операционных систем Windows приложение имеет динамически подключаемые библиотеки в комплекте и не требует установки никакого дополнительного программного обеспечения.

Работа в приложении осуществляется с помощью компьютерной мышки.

С помощью кнопок «-» и «+» осуществляется изменение размеров игрового поля. Выставленные размеры указаны слева от данных кнопок. При больших размерах игрового поля может потребоваться ожидание очередного хода компьютера, которое, в зависимости от оборудования, может составлять десятки секунд.

Ниже находится информация о состоянии игры в текущий момент.

Чекбокс «Первый ходит компьютер» позволяет дать возможность совершить первый ход (и дальнейшие) крестиком компьютеру, при этом игрок будет играть ноликом.

Кнопка «Перезапуск игры» сбрасывает игровое поле и состояние игры, применяет выставленные настройки к новой игровой партии.

В случае чьей-либо победы или заполнения игрового поля полностью дальнейшая простановка крестиков или ноликов на ней невозможна, партия считается завершённой и требуется начало новой партии с помощью кнопки перезапуска.

Выход из приложения осуществляется с помощью нажатия на стандартную иконку крестика вверху окна.

ЗАКЛЮЧЕНИЕ

Разработанное в данной курсовой работе приложение является готовым программным продуктом.

Приложение может быть использовано в исследовательских целях.

Стоит отметить, что реализованный алгоритм минимакс является самым простым, но неэффективным при большой глубине поиска. Используемое альфа-бета отсечение значительно уменьшает количество просматриваемых вариантов, но число операций всё же растёт экспоненциально.

В дальнейшем для улучшения производительности возможно дополнительное использование более сложных эвристических методов, как, например, итерационного углубления и таблицы перестановок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Structured binding declaration [Электронный ресурс] / cppreference.com – Режим доступа: https://en.cppreference.com/w/cpp/language/structured_binding. (Дата обращения: 08.02.2020)
2. Гончаров Е.Н. Исследование операций [Электронный ресурс]: учебное пособие / Гончаров Е.Н., Ерзин А.И., Залюбовский В.В. – Режим доступа: <http://math.nsc.ru/LBRT/k4/or/> – Новосибирск: НГУ, 2005. – 78 с. (Дата обращения: 30.01.2020)
3. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG [Текст] – 3-е изд. : Пер. с англ. – М.: Издательский дом "Вильямс", 2004. – 640 с.

ПРИЛОЖЕНИЕ 1

Файл main.cpp, содержащий глобальную функцию main

```
#include <SFML/Graphics.hpp>

#include "Utils/Assets.h"
#include "Game/Game.h"
#include "Game/Lookup.h"
#include "Gui/Gui.h"
#include "Config.h"

// Приведение "спроектированного" размера к размеру окна
sf::View CalcView(const sf::Vector2u& window_size,
                  const sf::Vector2u& designed_size) {
    sf::FloatRect viewport(0.f, 0.f, 1.f, 1.f);

    float screen_width = window_size.x / static_cast<float>(designed_size.x);
    float screen_height = window_size.y / static_cast<float>(designed_size.y);

    if (screen_width > screen_height) {
        viewport.width = screen_height / screen_width;
        viewport.left = (1.f - viewport.width) / 2.f;
    }
    else if (screen_width < screen_height) {
        viewport.height = screen_width / screen_height;
        viewport.top = (1.f - viewport.height) / 2.f;
    }

    sf::View view(sf::FloatRect(0, 0, designed_size.x, designed_size.y));
    view.setViewport(viewport);

    return view;
}

int main() {
    // Загрузка ресурсов
    Assets::Instance().Load();

    // Спроектированный размер
    const sf::Vector2u designed_size(kWindowWidth, kWindowHeight);
```

```

/* Создание окна игры */

// Сглаживание
sf::ContextSettings settings;
settings.antiAliasingLevel = 8;

sf::RenderWindow window(sf::VideoMode(kWindowWidth, kWindowHeight),
    "Tic-tac-toe", sf::Style::Default, settings);
window.setView(CalcView(window.getSize(), designed_size));
window.setFramerateLimit(60);

// Создаем объект игры
Lookup lookup;
Game game(lookup);
game.setPosition(kIndentFieldX, kIndentFieldY);

Gui gui(game);
gui.setPosition(kFieldSizePx + 2 * kIndentFieldX, kIndentFieldY);

while (window.isOpen()) {
    sf::Event event;
    while (window.pollEvent(event)) {
        sf::Vector2f point =
            window.mapPixelToCoords(sf::Mouse::getPosition(window));
        switch (event.type)
        {
            case sf::Event::Closed:
                window.close();
                break;
            case sf::Event::KeyPressed:
                if (event.key.code == sf::Keyboard::Escape) window.close();
                break;
            case sf::Event::Resized: {
                window.setView(
                    CalcView(sf::Vector2u(event.size.width, event.size.height),
                        designed_size)
                );
                break;
            }
            case sf::Event::MouseButtonPressed: {
                if (event.mouseButton.button == sf::Mouse::Left) {

```

```

        // Нажатие произошло внутри игрового поля
        if (game.IsPointInGameField(point))
            game.MouseButtonPressed(point);
        else // В противном случае - в гуи
            gui.MouseButtonPressed(point);
    }
    break;
}
case sf::Event::MouseButtonReleased: {
    if (event.mouseButton.button == sf::Mouse::Left &&
        !game.IsPointInGameField(point)) {
        gui.MouseButtonReleased(point);
    }
    break;
}
case sf::Event::MouseMove: {
    // Курсор перемещён внутри игрового поля
    if (game.IsPointInGameField(point))
        game.MouseMoved(point);
    else // В противном случае - в гуи
        gui.MouseMoved(point);
    break;
}
default:
    break;
}
}

// Обновление состояния
gui.Refresh();

// Отрисовка игрового поля и GUI
window.clear(sf::Color::White);
window.draw(game);
window.draw(gui);
window.draw(Assets::Instance().logo);
window.display();
}

return 0;
}

```

ПРИЛОЖЕНИЕ 2

Заголовочный файл Lookup.h, содержащий объявление класса Lookup

```
#pragma once

#include "Game.h"

#include <vector>

class Lookup {
protected:
    std::vector<std::vector<std::pair<int, int>>> winning_states_;
    std::vector<std::pair<int, int>> GetLegalMoves(
        std::vector<std::vector<StateCell>>& board);
    std::vector<std::pair<int, int>> GetOccupiedPositions(
        std::vector<std::vector<StateCell>>& board, StateCell marker);
    bool IsBoardFull(std::vector<std::vector<StateCell>>& board);
    std::pair<bool, std::vector<std::pair<int, int>>> IsGameWon(
        std::vector<std::vector<StateCell>>&,
        std::vector<std::pair<int, int>>& occupied_positions);
    StateCell GetOpponentMarker(StateCell marker);
    int GetBoardState(std::vector<std::vector<StateCell>>& board,
        StateCell marker);
public:
    Lookup() {};
    StateCell player_marker = StateCell::X, ai_marker = StateCell::O;
    void InitializeWinningStates(unsigned int count);
    bool IsGameDone(std::vector<std::vector<StateCell>>& board);
    std::pair<StateCell, std::vector<std::pair<int, int>>>
        GetWinningPosition(std::vector<std::vector<StateCell>>& board);
    std::pair<int, std::pair<int, int>> MinimaxOptimization(
        std::vector<std::vector<StateCell>>& board, StateCell marker, int depth,
        int alpha, int beta);
};
```

ПРИЛОЖЕНИЕ 3

Файл Lookup.cpp, содержащий реализацию класса Lookup

```
#include "Lookup.h"

#include "Game.h"
#include "Config.h"

// Заполнение выигрышных состояний
void Lookup::InitializeWinningStates(unsigned int count) {
    winning_states_.clear();
    // Строки и колонки
    for (int i = 0; i < count; i++) {
        std::vector<std::pair<int, int>> row;
        std::vector<std::pair<int, int>> column;
        for (int j = 0; j < count; j++) {
            row.push_back(std::make_pair(i, j));
            column.push_back(std::make_pair(j, i));
        }
        winning_states_.push_back(row);
        winning_states_.push_back(column);
    }

    // Диагонали
    std::vector<std::pair<int, int>> main_diagonal;
    std::vector<std::pair<int, int>> anti_diagonal;
    for (int i = 0; i < count; i++) {
        main_diagonal.push_back(std::make_pair(i, i));
        anti_diagonal.push_back(std::make_pair(count - i - 1, i));
    }
    winning_states_.push_back(main_diagonal);
    winning_states_.push_back(anti_diagonal);
}

// Получение всех свободных ячеек
std::vector<std::pair<int, int>> Lookup::GetLegalMoves(
    std::vector<std::vector<StateCell>>& board) {
    std::vector<std::pair<int, int>> legal_moves;
    for (int i = 0; i < board.size(); i++)
        for (int j = 0; j < board.size(); j++)
            if (board[i][j] != ai_marker && board[i][j] != player_marker)
```

```

        legal_moves.push_back(std::make_pair(i, j));

    return legal_moves;
}

// Получение всех ячеек, занятых крестиком или ноликом
std::vector<std::pair<int, int>> Lookup::GetOccupiedPositions(
    std::vector<std::vector<StateCell>>& board, StateCell marker) {
    std::vector<std::pair<int, int>> occupied_positions;

    for (int i = 0; i < board.size(); i++)
        for (int j = 0; j < board.size(); j++)
            if (marker == board[i][j])
                occupied_positions.push_back(std::make_pair(i, j));
    return occupied_positions;
}

// Проверка, что на доске нет больше мест
bool Lookup::IsBoardFull(std::vector<std::vector<StateCell>>& board) {
    std::vector<std::pair<int, int>> legal_moves = GetLegalMoves(board);
    return (0 == legal_moves.size());
}

// Проверка, что игра была выиграна
std::pair<bool, std::vector<std::pair<int, int>>> Lookup::IsGameWon(
    std::vector<std::vector<StateCell>>& board,
    std::vector<std::pair<int, int>>& occupied_positions) {
    bool game_won;

    std::vector<std::pair<int, int>> curr_win_state;
    for (int i = 0; i < winning_states_.size(); i++) {
        game_won = true;
        curr_win_state = winning_states_[i];
        for (int j = 0; j < board.size(); j++) {
            // Если ни одна выигрышная позиция не найдена, игра "не выиграна"
            auto iter = std::find(
                std::begin(occupied_positions),
                std::end(occupied_positions),
                curr_win_state[j]
            );
            if (iter == std::end(occupied_positions)) {
                game_won = false;
            }
        }
    }
}

```



```

        break;
    }
}

    if (game_won) break;
}

return std::make_pair(game_won, curr_win_state);
}

StateCell Lookup::GetOpponentMarker(StateCell marker) {
    StateCell opponent_marker;
    if (marker == player_marker)
        opponent_marker = ai_marker;
    else
        opponent_marker = player_marker;

    return opponent_marker;
}

// Проверка на выигрыш или проигрыш
int Lookup::GetBoardState(std::vector<std::vector<StateCell>>& board,
                          StateCell marker) {
    StateCell opponent_marker = GetOpponentMarker(marker);

    std::vector<std::pair<int, int>> occupied_positions =
        GetOccupiedPositions(board, marker);

    bool is_won = IsGameWon(board, occupied_positions).first;
    if (is_won) return kWin;

    occupied_positions = GetOccupiedPositions(board, opponent_marker);

    bool is_lost = IsGameWon(board, occupied_positions).first;
    if (is_lost) return kLoss;
    bool is_full = IsBoardFull(board);
    if (is_full) return kDraw;
    return kDraw;
}

// Минимакс алгоритм
std::pair<int, std::pair<int, int>> Lookup::MinimaxOptimization(

```

```

std::vector<std::vector<StateCell>>& board, StateCell marker, int depth,
int alpha, int beta) {
// Инициализация лучшего хода
std::pair<int, int> best_move = std::make_pair(-1, -1);
int best_score = (marker == ai_marker) ? kLoss : kWin;

// Если достигнут конец дерева, возврат лучшего результата и хода
int board_state = GetBoardState(board, ai_marker);
if (IsBoardFull(board) || kDraw != board_state || depth >= kMaxDepth)
return std::make_pair(board_state, best_move);

std::vector<std::pair<int, int>> legal_moves = GetLegalMoves(board);
for (int i = 0; i < legal_moves.size(); i++) {
std::pair<int, int> curr_move = legal_moves[i];
board[curr_move.first][curr_move.second] = marker;

// Максимизация стратегии игрока
if (marker == ai_marker) {
int score =
MinimaxOptimization(board, player_marker, depth + 1, alpha, beta).first;

if (best_score < score) {
best_score = score - depth * 10;
best_move = curr_move;

// Если лучший ход этой ветви хуже, чем лучший предыдущей,
// то откидываем её
alpha = std::max(alpha, best_score);
board[curr_move.first][curr_move.second] = StateCell::None;
if (beta <= alpha) break;
}

} // Минимизация стратегии бота
else {
int score =
MinimaxOptimization(board, ai_marker, depth + 1, alpha, beta).first;

if (best_score > score) {
best_score = score + depth * 10;
best_move = curr_move;

// Если лучший ход этой ветви хуже, чем лучший предыдущей,

```

```

        // то откидываем её
        beta = std::min(beta, best_score);
        board[curr_move.first][curr_move.second] = StateCell::None;
        if (beta <= alpha) break;
    }

}

board[curr_move.first][curr_move.second] = StateCell::None; // Отмена хода

}

return std::make_pair(best_score, best_move);
}

// Проверка, что игра завершена
bool Lookup::IsGameDone(std::vector<std::vector<StateCell>>& board) {
    if (IsBoardFull(board))
        return true;
    if (kDraw != GetBoardState(board, ai_marker))
        return true;

    return false;
}

std::pair<StateCell, std::vector<std::pair<int, int>>>
    Lookup::GetWinningPosition(std::vector<std::vector<StateCell>>& board) {
    std::vector<std::pair<int, int>> occupied_positions =
        GetOccupiedPositions(board, player_marker);

    std::pair<bool, std::vector<std::pair<int, int>>> player_won =
        IsGameWon(board, occupied_positions);
    if (player_won.first) return std::make_pair(player_marker, player_won.second);

    occupied_positions = GetOccupiedPositions(board, ai_marker);

    std::pair<bool, std::vector<std::pair<int, int>>> ai_won =
        IsGameWon(board, occupied_positions);
    if (ai_won.first) return std::make_pair(ai_marker, ai_won.second);

    return std::make_pair(StateCell::None, std::vector<std::pair<int, int>>());
}

```

ПРИЛОЖЕНИЕ 4

Заголовочный файл Game.h, содержащий объявление класса Game

```
#pragma once

#include <SFML/Graphics.hpp>
#include "Config.h"

class Lookup;

enum class StateCell { None = 0, X = 1, O = 2 };

class Game : public sf::Drawable, public sf::Transformable {
private:
    Lookup& lookup_;
    unsigned int cells_count_;
    float cell_size_;
    std::pair<StateCell, std::vector<std::pair<int, int>>> win_state_;
    bool is_first_turn_ai_ = false;
public:
    Game(Lookup& l);
    std::vector<std::vector<StateCell>> board;
    void SetFirstTurnAI(bool state) { is_first_turn_ai_ = state; }
    bool GetFirstTurnAI() { return is_first_turn_ai_; }
    void ResizeBoard();
    void MouseButtonPressed(sf::Vector2f point);
    void MouseMoved(sf::Vector2f point) {};
    void SetCellsCount(unsigned int count) { cells_count_ = count; }
    unsigned int GetCellsCount() { return cells_count_; }
    bool IsFinished();
    StateCell GetWinner() { return win_state_.first; }
    bool IsPointInGameField(sf::Vector2f point);
public:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
};
```

ПРИЛОЖЕНИЕ 5

Файл Game.cpp, содержащий реализацию класса Game

```
#include "Game.h"

#include "Lookup.h"
#include "Config.h"

#include <math.h>

Game::Game(Lookup& l) : lookup_(l) {
    SetCellsCount(kDefaultFieldSize);
    ResizeBoard();
}

bool Game::IsPointInGameField(sf::Vector2f point) {
    sf::Vector2f point_in_container = point - getPosition();
    return (point_in_container.x >= 0 && point_in_container.x <= kFieldSizePx) &&
        (point_in_container.y >= 0 && point_in_container.y <= kFieldSizePx);
}

bool Game::IsFinished() {
    return lookup_.IsGameDone(board);
}

void Game::ResizeBoard() {
    cell_size_ = kFieldSizePx / cells_count_;
    lookup_.InitializeWinningStates(cells_count_);
    win_state_ =
        std::make_pair(StateCell::None, std::vector<std::pair<int, int>>());
    board.clear();
    board.resize(cells_count_);
    for (int i = 0; i < cells_count_; ++i)
        board[i].resize(cells_count_);

    if (is_first_turn_ai_) {
        lookup_.ai_marker = StateCell::X;
        lookup_.player_marker = StateCell::O;
        board[cells_count_ / 2][cells_count_ / 2] = lookup_.ai_marker;
    } else {
        lookup_.ai_marker = StateCell::O;
    }
}
```

```

        lookup_.player_marker = StateCell::X;
    }
}

void Game::MouseButtonPressed(sf::Vector2f point) {
    if (IsFinished()) return;

    sf::Vector2f point_in_container = point - getPosition();
    // Вычисление текущей ячейки
    unsigned int column = point_in_container.x / cell_size_;
    unsigned int row = point_in_container.y / cell_size_;

    // Изменение статуса ячейки
    if ((row < cells_count_ && column < cells_count_) &&
        (board[row][column] == StateCell::None)) {

        board[row][column] = lookup_.player_marker;

        std::pair<int, std::pair<int, int>> ai_move =
            lookup_.MinimaxOptimization(board, lookup_.ai_marker, 0, kLoss, kWin);

        if (ai_move.second.first != -1 && ai_move.second.second != -1)
            board[ai_move.second.first][ai_move.second.second] = lookup_.ai_marker;

        win_state_ = lookup_.GetWinningPosition(board);
    }
}

void Game::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    states.transform *= getTransform();

    /* Отрисовка игрового поля */

    for (unsigned int i = 1; i < board.size(); i++) {
        // Горизонтальная линия
        sf::Vector2f position(0, cell_size_ * i - kLineWidth / 2);

        sf::RectangleShape line(
            sf::Vector2f(cell_size_ * board.size(), kLineWidth)
        );
        line.setFillColor(sf::Color::Black);
        line.setPosition(position);
    }
}

```

```

target.draw(line, states);

// Вертикальная линия
line.rotate(90);

position.x = position.y;
position.y = 0;
line.setPosition(position);

target.draw(line, states);
}

/* Отрисовка состояний ячеек */

for (unsigned int i = 0; i < board.size(); i++) {
    for (unsigned int j = 0; j < board.size(); j++) {
        switch (board[i][j]) {
            case StateCell::X: {
                // Вычисление смещений по координатам
                float cathet = cell_size_ / sqrt(2);
                float indent = cell_size_ / 2 - cathet / 2;
                sf::RectangleShape line(sf::Vector2f(cell_size_, kMarkerWidth));
                line.setFillColor(sf::Color::Blue);
                line.rotate(45);
                line.setPosition(
                    sf::Vector2f(cell_size_ * j + indent,
                                cell_size_ * i + indent - kMarkerWidth / 2)
                );
                target.draw(line, states);
                line.rotate(90);
                line.setPosition(
                    sf::Vector2f(cell_size_ * j + cathet + indent,
                                cell_size_ * i + indent + kMarkerWidth / 2)
                );
                target.draw(line, states);
                break;
            }
            case StateCell::O: {
                float radius = cell_size_ / M_PI;
                sf::CircleShape circle;
                circle.setRadius(radius);

```

```

        circle.setOutlineColor(sf::Color::Red);
        circle.setOutlineThickness(kMarkerWidth);
        circle.setPosition(
            sf::Vector2f(cell_size_ * j + radius / 2,
                        cell_size_ * i + radius / 2)
        );
        target.draw(circle, states);
        break;
    }
}
}
}

/* Отрисовка победных состояний */

if (win_state_.first != StateCell::None) {
    // Преобразование победного состояния в координаты
    sf::Vector2f position_begin(
        cell_size_ * win_state_.second[0].second,
        cell_size_ * win_state_.second[0].first
    );
    sf::Vector2f position_end(
        cell_size_ * (win_state_.second[board.size() - 1].second + 1),
        cell_size_ * (win_state_.second[board.size() - 1].first + 1)
    );

    // Смещение "зачёркивания" для трёх вариантов: строка, колонка,
    // диагональ
    if (win_state_.second[0].second ==
        win_state_.second[board.size() - 1].second) { // Колонка
        position_begin += sf::Vector2f(cell_size_ / 2 + kMarkerWidth / 4, 0);
        position_end -= sf::Vector2f(cell_size_ / 2 - kMarkerWidth / 4, 0);
    } else if (win_state_.second[0].first ==
        win_state_.second[board.size() - 1].first) { // Строка
        position_begin += sf::Vector2f(0, cell_size_ / 2 - kMarkerWidth / 2);
        position_end -= sf::Vector2f(0, cell_size_ / 2 + kMarkerWidth / 2);
    } else if (win_state_.second[0].first ==
        win_state_.second[board.size() - 1].second) { // Побочная диагональ
        position_begin += sf::Vector2f(0, cell_size_ - kMarkerWidth);
        position_end -= sf::Vector2f(0, cell_size_ + kMarkerWidth);
    } else { // Главная диагональ
        position_begin -= sf::Vector2f(0, kMarkerWidth / 2);
    }
}

```



```

    position_end -= sf::Vector2f(0, kMarkerWidth / 2);
}

sf::RectangleShape line(
    sf::Vector2f(
        sqrt(
            pow(position_end.x - position_begin.x, 2) +
            pow(position_end.y - position_begin.y, 2)
        ),
        kMarkerWidth)
);

// Цвет в зависимости от победивших фигур
line.setFillColor(
    win_state_.first == StateCell::O ? sf::Color::Red : sf::Color::Blue
);

line.setPosition(position_begin);

// Поворот "зачёркивания" на соответствующий угол
float rad = atan2(position_end.y - position_begin.y,
    position_end.x - position_begin.x);

line.rotate(180 / M_PI * rad);

target.draw(line, states);
}
}

```

ПРИЛОЖЕНИЕ 6

Заголовочный файл Gui.h, содержащий объявление класса Gui

```
#pragma once

#include <SFML/Graphics.hpp>

#include <memory>

// Предварительное объявление
class Game;
class Control;

class Gui : public sf::Drawable, public sf::Transformable {
private:
    Game& game_;
    std::map<std::string, std::shared_ptr<Control>> controls_;
    bool IsPointInside(const std::shared_ptr<Control>& control,
        sf::Vector2f point);
public:
    Gui(Game& game);
    void Initialize();
    void MouseButtonPressed(sf::Vector2f point);
    void MouseButtonReleased(sf::Vector2f point);
    void MouseMoved(sf::Vector2f point);
    void Refresh();
public:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
};
```

ПРИЛОЖЕНИЕ 7

Файл Gui.cpp, содержащий реализацию класса Gui

```
#include "Gui.h"

#include "Game/Game.h"

#include "Control.h"
#include "Button.h"
#include "Label.h"
#include "Checkbox.h"
#include "Config.h"

#include <iostream>
#include <string>

Gui::Gui(Game& g) : game_(g) {
    Initialize();
}

void Gui::Initialize() {
    auto label_start = std::make_shared<Label>(
        std::wstring(L"Игра \"Крестики-нолики\"\\n") +
        std::wstring(L"Первый ход делает игрок, ставящий крестики.")
    );

    /* Кнопка сброса на стандартные настройки (Config.h) */
    auto btn_default_config = std::make_shared<Button>(
        L"Стандартный размер поля", sf::Vector2f(0, 50)
    );
    btn_default_config->SetSize(sf::Vector2f(350, kButtonDefaultHeight));
    btn_default_config->OnClick = [](Control& me, Game& game, Gui& gui) {
        game.SetCellsCount(kDefaultFieldSize);
    };

    /* Лейбл для вывода размерности поля */
    auto label_field_size = std::make_shared<Label>(sf::Vector2f(0, 100));

    /* Кнопка увеличения размерности поля */
    auto btn_reduce_field = std::make_shared<Button>(L"-",
        sf::Vector2f(240, label_field_size->GetPosition().y - 5)
```

```

);
btn_reduce_field->SetSize(sf::Vector2f(50, kButtonDefaultHeight));
btn_reduce_field->OnClick = [](Control& me, Game& game, Gui& gui) {
    if (game.GetCellsCount() > 3)
        game.SetCellsCount(game.GetCellsCount() - 1);
};

/* Кнопка уменьшения размерности поля */
auto btn_increase_field = std::make_shared<Button>(L"+",
    sf::Vector2f(300, label_field_size->GetPosition().y - 5)
);
btn_increase_field->SetSize(sf::Vector2f(50, kButtonDefaultHeight));
btn_increase_field->OnClick = [](Control& me, Game& game, Gui& gui) {
    if (game.GetCellsCount() < 10)
        game.SetCellsCount(game.GetCellsCount() + 1);
};

/* Лейбл для вывода состояния игры */
auto label_status_game = std::make_shared<Label>(L"", sf::Vector2f(0, 150));

/* Чекбокс, передающий право первого хода компьютеру */
auto checkbox_first_ai = std::make_shared<Checkbox>(
    L"Первый ходит компьютер",
    sf::Vector2f(0, 200)
);
checkbox_first_ai->OnClick = [](Control& me, Game& game, Gui& gui) {
    Checkbox& me_casted = static_cast<Checkbox&>(me);
    bool is_checked = me_casted.GetCheckedState();
    me_casted.SetCheckedState(!is_checked);
    game.SetFirstTurnAI(!is_checked);
};

/* Кнопка перезапуска игры */
auto btn_restart_game = std::make_shared<Button>(
    L"Перезапуск игры", sf::Vector2f(0, 250)
);
btn_restart_game->SetSize(sf::Vector2f(350, kButtonDefaultHeight));
btn_restart_game->OnClick = [](Control& me, Game& game, Gui& gui) {
    game.ResizeBoard();
};

controls_["label_start"] = label_start;

```

```

controls_["btn_default_config"] = btn_default_config;
controls_["label_field_size"] = label_field_size;
controls_["btn_reduce_field"] = btn_reduce_field;
controls_["btn_increase_field"] = btn_increase_field;
controls_["label_status_game"] = label_status_game;
controls_["checkbox_first_ai"] = checkbox_first_ai;
controls_["btn_restart_game"] = btn_restart_game;
}

void Gui::Refresh() {
    /* Обновление размерности поля */
    controls_.find("label_field_size")->second->SetTitleText(
        L"Размер игрового поля: " +
        std::to_wstring(game_.GetCellsCount()) + L"x" +
        std::to_wstring(game_.GetCellsCount())
    );

    /* Обновление состояния игры */
    std::wstring state_of_game_label;
    if (game_.IsFinished()) {
        if (game_.GetWinner() == StateCell::X)
            state_of_game_label = L"Выиграли крестики";
        else if (game_.GetWinner() == StateCell::O)
            state_of_game_label = L"Выиграли нолики";
        else
            state_of_game_label = L"Ничья";
    } else {
        state_of_game_label = L"Игра продолжается";
    }

    controls_.find("label_status_game")->second->SetTitleText(
        L"Состояние игры: " + state_of_game_label
    );
}

void Gui::MouseButtonPressed(sf::Vector2f point) {
    for (const auto& [name, control]: controls_)
        control->SetPressed(IsPointInside(control, point));
}

void Gui::MouseButtonReleased(sf::Vector2f point) {
    for (auto const& [name, control]: controls_) {

```

```

        if (IsPointInside(control, point) &&
            (control->GetPressed() && control->OnClick))
            control->OnClick(*control, game_, *this);

        control->SetPressed(false);
    }
}

void Gui::MouseMoved(sf::Vector2f point) {
    for (auto const& [name, control]: controls_)
        control->SetHovered(IsPointInside(control, point));
}

bool Gui::IsPointInside(const std::shared_ptr<Control>& control,
                        sf::Vector2f point) {
    // Преобразование в координаты внутри GUI
    sf::Vector2f point_in_container = point - getPosition();
    // Получение границ элемента
    float left = control->GetPosition().x;
    float right = left + control->GetSize().x;
    float top = control->GetPosition().y;
    float bottom = top + control->GetSize().y;

    return (left <= point_in_container.x && point_in_container.x <= right) &&
        (top <= point_in_container.y && point_in_container.y <= bottom);
}

void Gui::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    states.transform *= getTransform();

    // Отрисовка элементов GUI
    for (auto const& [name, control]: controls_)
        target.draw(*control, states);
}

```

ПРИЛОЖЕНИЕ 8

Заголовочный файл Control.h, содержащий объявление класса Control

```
#pragma once

#include <SFML/Graphics.hpp>

class Game;
class Gui;

class Control : public sf::Drawable, public sf::Transformable {
protected:
    std::wstring title_text_;
    bool is_hovered_ = false;
    bool is_pressed_ = false;
    sf::Vector2f size_;
    sf::Vector2f position_;
public:
    Control(std::wstring title_text, sf::Vector2f position) {
        title_text_ = title_text;
        position_ = position;
    };
    void (*OnClick) (Control& me, Game& game, Gui& gui) = NULL;
    sf::Vector2f GetSize() { return size_; }
    sf::Vector2f GetPosition() { return position_; }
    bool GetPressed() { return is_pressed_; }
    void SetSize(sf::Vector2f val) { size_ = val; }
    void SetHovered(bool state) { is_hovered_ = state; }
    void SetPressed(bool state) { is_pressed_ = state; }
    void SetTitleText(std::wstring title_text) {
        title_text_ = title_text;
    }
};
```

ПРИЛОЖЕНИЕ 9

Заголовочный файл Assets.h, содержащий объявление класса Assets

```
#pragma once

#include <SFML/Graphics.hpp>

// Синглтон Майерса
class Assets {
private:
    sf::Texture logo_texture_;
public:
    sf::Font font;
    sf::Sprite logo;
public:
    static Assets& Instance() {
        static Assets s;
        return s;
    }
    void Load();
private:
    Assets() {};
    ~Assets() {};
    Assets(Assets const&) = delete;
    Assets& operator= (Assets const&) = delete;
};
```


ПРИЛОЖЕНИЕ 10

Файл Assets.cpp, содержащий реализацию класса Assets

```
#include "Assets.h"

#include "Config.h"

void Assets::Load() {
    if (!font.loadFromFile(kFontPath)) throw;
    if (!logo_texture_.loadFromFile(kLogoPath)) throw;
    logo_texture_.setSmooth(true); // Сглаживание текстуры
    logo.setTexture(logo_texture_);
    logo.setPosition(
        sf::Vector2f(
            kIndentFieldX + kFieldSizePx + kIndentFieldX,
            kWindowHeight - 200
        )
    );
    logo.scale(0.8f, 0.8f); // Сжатие по сторонам
}
```

ПРИЛОЖЕНИЕ 11

Заголовочный файл Label.h, содержащий объявление класса Label

```
#pragma once

#include "Control.h"

#include "Config.h"

class Label : public Control {
public:
    Label(std::wstring title_text) : Label(title_text, sf::Vector2f()) {};
    Label(sf::Vector2f position) : Label(std::wstring(), position) {};
    Label(std::wstring title_text, sf::Vector2f position) :
        Control(title_text, position) {};
    ~Label() {};
public:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
};
```

ПРИЛОЖЕНИЕ 12

Файл Label.cpp, содержащий реализацию класса Label

```
#include "Label.h"

#include "Utils/Assets.h"

void Label::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    states.transform *= getTransform();

    sf::Text text(title_text_, Assets::Instance().font, kDefaultFontSize);
    text.setFillColor(sf::Color::Black);

    text.setPosition(position_);

    target.draw(text, states);
}
```

ПРИЛОЖЕНИЕ 13

Заголовочный файл Button.h, содержащий объявление класса Button

```
#pragma once

#include "Control.h"

#include "Config.h"

class Button : public Control {
public:
    Button(std::wstring title_text) : Button(title_text, sf::Vector2f()) { };
    Button(std::wstring title_text, sf::Vector2f position) :
        Control(title_text, position) {
        size_ = sf::Vector2f(kButtonDefaultWidth, kButtonDefaultHeight);
    }
    ~Button() {};
public:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
};
```

ПРИЛОЖЕНИЕ 14

Файл Button.cpp, содержащий реализацию класса Button

```
#include "Button.h"

#include "Utils/Assets.h"

void Button::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    states.transform *= getTransform();

    /* Отрисовка контура кнопки */

    sf::RectangleShape shape(sf::Vector2f(size_.x, size_.y));

    // Выбор цвета в зависимости от состояния
    if (is_pressed_)
        shape.setFillColor(sf::Color(230, 130, 60));
    else if (is_hovered_)
        shape.setFillColor(sf::Color(50, 50, 50));
    else
        shape.setFillColor(sf::Color(100, 100, 100));

    shape.setPosition(position_);

    target.draw(shape, states);

    /* Отрисовка текста внутри кнопки */

    sf::Text text(title_text_, Assets::Instance().font, kDefaultFontSize);
    text.setFillColor(sf::Color::White);

    // Вычисление позиции для текста
    int title_x = shape.getPosition().x - text.getLocalBounds().left +
        shape.getSize().x / 2 - text.getGlobalBounds().width / 2,
        title_y = shape.getPosition().y - text.getLocalBounds().top +
        shape.getSize().y / 2 - text.getGlobalBounds().height / 2;

    text.setPosition(title_x, title_y);

    target.draw(text, states);
}
```

ПРИЛОЖЕНИЕ 15

Файл Checkbox.h, содержащий объявление класса Checkbox

```
#pragma once

#include "Control.h"

#include "Config.h"

class Checkbox : public Control {
private:
    bool is_checked_ = false;
public:
    Checkbox(std::wstring title_text) : Checkbox(title_text, sf::Vector2f()) { };
    Checkbox(std::wstring title_text, sf::Vector2f position) :
        Control(title_text, position) {
        size_ = sf::Vector2f(kCheckboxDefaultWidth, kCheckboxDefaultHeight);
    }
    void SetCheckedState(bool state) { is_checked_ = state; }
    bool GetCheckedState() { return is_checked_; }
    ~Checkbox() {};
public:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
};
```

ПРИЛОЖЕНИЕ 16

Файл Checkbox.cpp, содержащий реализацию класса Checkbox

```
#include "Checkbox.h"
#include "Utils/Assets.h"

void Checkbox::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    states.transform *= getTransform();

    /* Отрисовка контура "флажка" */
    sf::RectangleShape shape(size_);
    shape.setOutlineThickness(kLineWidth);
    shape.setOutlineColor(sf::Color::Black);
    shape.setPosition(position_);
    target.draw(shape, states);

    /* Отрисовка "флажка" */
    sf::Vector2f indent(6, 6); // Отступ от границ контура
    sf::RectangleShape flag(size_ - indent);
    // Выбор цвета в зависимости от состояния
    if (is_checked_)
        flag.setFillColor(sf::Color(230, 130, 60));
    else if (is_hovered_)
        flag.setFillColor(sf::Color(50, 50, 50));
    flag.setPosition(
        sf::Vector2f(position_.x + indent.x / 2, position_.y + indent.y / 2)
    );
    target.draw(flag, states);
    /* Отрисовка текста слева */
    sf::Text text(title_text_, Assets::Instance().font, kDefaultFontSize);
    text.setFillColor(sf::Color::Black);

    // Вычисление позиции для текста
    int title_x = shape.getSize().x + 10,
        title_y = shape.getPosition().y - text.getLocalBounds().top +
            shape.getSize().y / 2 - text.getGlobalBounds().height / 2;

    text.setPosition(title_x, title_y);

    target.draw(text, states);
}
```

ПРИЛОЖЕНИЕ 17

Файл Config.h, содержащий основные константы приложения

```
#pragma once

// Размеры окна
const int kWindowWidth = 1000;
const int kWindowHeight = 600;

const int kDefaultFieldSize = 3; // Размер игрового поля в квадратах
const int kFieldSizePx = 500; // Размер игрового поля в пикселях
const int kLineWidth = 2; // Толщина линий игрового поля
const int kMarkerWidth = 5; // Толщина крестиков и ноликов

const int kIndentFieldX = 50; // Отступ перед игровым полем по X в пикселях
const int kIndentFieldY = 50; // Отступ перед игровым полем по Y в пикселях

// Ресурсы игры
const std::string kFontPath = "resources/calibri.ttf";
const std::string kLogoPath = "resources/logo.png";
const int kDefaultFontSize = 17; // Размер шрифта

// Стандартные размеры элементов
const int kButtonDefaultWidth = 100;
const int kButtonDefaultHeight = 30;
const int kCheckboxDefaultWidth = 20;
const int kCheckboxDefaultHeight = 20;

const int kMaxDepth = 4; // Максимальная глубина минимакса
const int kWin = 1000; // Значение победы
const int kDraw = 0; // Значение ничьей
const int kLoss = -1000; // Значение поражения
```