

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Пензенский государственный университет»
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по курсу «Программирование»
на тему «Разработка программы сложной структуры методом нисходящего
программирования. Игра Крестики-Нолики»

Отлично
22.05.26
Г. Генералова

Выполнил:

студент группы 25BVB2

Ионова А. Н.

Принял:

к.т.н., доцент Генералова А.А.

Пенза 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

М.А. Митрохин

« 11 » февраля 2026 г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Новиков Иван Николаевич Группа 258882

Тема проекта: **Разработка программы сложной структуры методом нисходящего программирования.**

Исходные данные (технические требования) на проектирование

*Игра "Крестики-нолики". Игра первого хода
назначается пользователю. Компьютер генерирует
поле для игры. Пользователь пытается найти выигрыш
ход для победы. В файл записывается и сохраняется
таблица рекордов.*

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания " 11 " февраля 2026

Дата защиты проекта " 22 " апреля 2026

Руководитель Г. Генералов А.А.

Задание получил " 11 " февраля 2026г.

Студент А. Мухоморов А.А.

Содержание

Введение	5
Постановка задачи	5
Выбор решения	7
<i>Определение необходимых модулей программы</i>	7
Описание разработки программы	10
Отладка и тестирование	12
Описание программы	13
Руководство пользователя	22
Заключение	23
Список литературы	25
Приложение А	26
Листинги программы	26
<i>game.cpp</i>	26
<i>game.h</i>	54
<i>mainn.cpp</i>	54

Введение

Крестики-нолики представляют собой логическую игру между двумя противниками на квадратном поле. Классический вариант игры использует поле размером 3 на 3 клетки, однако существуют модификации вплоть до бесконечного поля. Для каждой из сторон общеизвестны алгоритмы, которые гарантируют ничью при любой игре противника, а при его ошибке позволяют выиграть.

Актуальность и востребованность реализации данной игры программным путём обусловлена необходимостью изучения базовых принципов искусственного интеллекта, алгоритмов защиты и атаки, а также отработки навыков структурного программирования. В условиях роста сложности разрабатываемых программных продуктов понимание логики подобных игр позволяет перейти к более сложным системам, таким как шахматы или го, где также применяются эвристические правила.

В наше время игровые алгоритмы, заложенные в крестики-нолики, используются не только в развлекательных целях. Они находят применение в обучении систем принятия решений, в моделировании конфликтных ситуаций и даже в некоторых задачах управления роботами, где требуется выбор оптимального хода в условиях ограниченного поля возможностей. Создавая игровую программу, программист стремится реализовать чёткие приоритетные правила, гарантирующие ничью при идеальной игре противника.

Считается, что игрок всегда соблюдает два правила, имеющих приоритет над всеми остальными. Правило 1: если игрок может немедленно выиграть, он это делает. Правило 2: если игрок не может немедленно выиграть, но его противник мог бы немедленно выиграть, сделав ход в какую-то клетку, игрок сам делает ход в эту клетку, предотвращая немедленный проигрыш. Эти правила лежат в основе минимаксной стратегии и обеспечивают идеальную игру.

В процессе выполнения курсовой работы была разработана программа на языке C++ (или с использованием ассемблерных вставок), позволяющая выполнять полноценную игровую сессию между двумя игроками. Программа

обеспечивает отображение игрового поля, контроль корректности ходов, автоматическое определение победы или ничьей, а также сохранение результатов игр (победитель, ничья, дата и время) в файл и последующий вывод истории партий на экран. Такая реализация позволяет не только играть вдвоём, но и анализировать статистику встреч, а также наглядно демонстрировать влияние правил атаки и защиты на исход игры.

Постановка задачи

Необходимо разработать программу «Крестики-нолики». Информация в игре структурирована по определённым признакам: игровое поле размером 3×3 клетки, два игрока (человек и компьютер), поочерёдность ходов, условия победы (три символа в ряд по горизонтали, вертикали или диагонали), условие ничьей (заполнение всех клеток без победной комбинации), а также статистика игроков, включающая имя игрока, количество побед, ничьих и поражений. Программа должна предоставить возможности: выхода из программы, начала новой игры, ввода имени игрока, поочерёдного выполнения ходов человеком и компьютером, автоматического определения победителя или ничьей, отображения текущего состояния игрового поля в псевдографическом виде, сохранения статистики игроков в файл, загрузки статистики из файла, отображения таблицы рекордов с сортировкой по количеству побед, а также использования ассемблерных вставок для оптимизации проверки победных комбинаций.

Многомодульность программы. В представленном коде реализована логическая модульность: функция инициализации поля (`initializeBoard`), функция отображения поля (`printBoard`), функция проверки победителя с ассемблерной вставкой (`checkWinner`), функция проверки заполненности поля (`isBoardFull`), функция проверки корректности хода (`isValidMove`), функция оценки состояния поля для минимакса (`evaluateBoard`), рекурсивная функция минимакса (`minimax`), функция хода компьютера (`computerMove`), функции загрузки и сохранения статистики (`loadRecords`, `saveRecords`), функция поиска или создания записи об игроке (`findOrCreateRecord`), функция отображения таблицы рекордов (`showLeaderboard`), а также главная функция `main` с игровым циклом. Это повысит локализацию ошибок при отладке и тестировании программы.

Необходимо использовать сложные типы данных – двумерный массив `char board[3][3]` для представления игрового поля, структуру `Record` для хранения информации об игроке (имя, победы, ничьи, поражения), вектор `vector<Record>` для динамического хранения записей всех игроков, строки `string` для работы с именами, а также файловые потоки `ifstream` и `ofstream` для работы с

файлом records.txt. Это необходимо для реализации алгоритмов хранения состояния игры, статистики игроков и взаимодействия между модулями.

Устройство ввода информации – клавиатура. В программе реализован ввод имени игрока (строка), ввод номера строки и столбца для выполнения хода (два целых числа), а также ввод ответа на вопрос о повторной игре (символ 'y' или 'n'). Реализована обработка некорректного ввода через проверку потока cin, его очистку при ошибке и игнорирование остатка строки. Необходимо однозначно идентифицировать и выполнять те или иные действия в зависимости от действий пользователя. Это необходимо для лёгкого использования программы.

Пользовательский интерфейс должен быть построен на основе меню. В программе реализован циклический интерфейс с приглашениями к вводу: запрос имени игрока при запуске, отображение игрового поля и приглашение к ходу, отображение таблицы рекордов после каждой партии, запрос на повторную игру. Заставка отсутствует в текущей версии, но может быть легко добавлена перед началом основного цикла. Это необходимо для создания интуитивно понятного интерфейса.

Сохранение в файле создано для удобства. В программе реализовано сохранение статистики всех игроков в текстовый файл records.txt. Каждая запись содержит имя игрока (латиницей), количество побед, количество ничьих и количество поражений. При запуске программы данные загружаются из файла, а после каждой партии файл обновляется. С его помощью пользователь может сохранить свою историю игр, передать файл другому лицу для сравнения результатов или просто вести статистику побед и поражений на протяжении нескольких игровых сессий.

Выбор решения

Определение необходимых модулей программы

Файлы: «main.cpp», «game.cpp», «game.h» - необходимы для организации многомодульной структуры программы. Данные файлы содержат функции выхода из программы, начала новой игры, ввода ходов игроком, автоматического хода компьютера, проверки победы (с использованием ассемблерной вставки), проверки заполненности поля, отображения игрового поля, загрузки и сохранения статистики игроков, отображения таблицы рекордов.

Приложение представляет собой совокупность данных и модулей. Приложение использует данные, содержащиеся в файле «records.txt», где хранится статистика игроков (имя, победы, ничьи, поражения). Все вышеперечисленное удовлетворяет требованию модульности программы данной курсовой работы. Управление приложением осуществляется с помощью клавиатуры. Программа предназначена для игры в «Крестики-нолики» между человеком и компьютером, причём компьютер использует алгоритм минимакса для обеспечения оптимальной игры. Открытие и сохранение выходного файла выполняется средствами данной программы автоматически: при запуске программа загружает статистику из файла «records.txt», а после каждой партии сохраняет обновлённую статистику в этот же файл. Программа даёт возможность просматривать таблицу рекордов, играть партии, наблюдать за ходом компьютера, видеть текущее состояние игрового поля, получать информацию о победителе или ничьей. Данные из файла статистики отображаются в программе в виде таблицы, содержащей столбцы: имя игрока, количество побед, количество ничьих, количество поражений, отсортированные по убыванию побед.

При запуске программы открывается приветственный экран с информацией о курсовой работе. Далее начинается игровой процесс, организованный в виде цикла, состоящего из следующих этапов:

1. Выход из программы. Пользователь нажимает 'n' при запросе «Сыграем ещё?», после чего программа завершает работу, сохраняя всю статистику в файл.

2. Начало новой игры. Поле инициализируется пустыми клетками, игрок получает символ 'X', компьютер — символ 'O', первый ход всегда принадлежит игроку.
3. Выполнение хода игроком. Пользователь вводит номер строки и столбца (от 1 до 3) для установки своего символа. Программа проверяет корректность хода: клетка должна быть пустой, а координаты — находиться в пределах поля. При неверном вводе программа запрашивает координаты повторно.
4. Автоматический ход компьютера. Компьютер использует алгоритм минимакса (minimax) для выбора оптимального хода. Программа перебирает все возможные варианты, оценивает каждый ход и выбирает наилучший с точки зрения максимизации своей победы и минимизации победы игрока.
5. Проверка победы. После каждого хода программа с помощью вставки на языке ассемблера проверяет все восемь победных комбинаций (три строки, три столбца, две диагонали). Если три одинаковых символа выстроены в ряд, программа объявляет победителя.
6. Проверка ничьей. Если поле полностью заполнено, но победитель не выявлен, программа объявляет ничью.
7. Обновление статистики. В зависимости от исхода партии программа увеличивает счётчик побед, ничьих или поражений для текущего игрока в структуре Record.
8. Отображение таблицы рекордов. После завершения каждой партии программа выводит на экран таблицу рекордов, в которой все игроки отсортированы по количеству побед (от большего к меньшему). Таблица содержит имя игрока, количество побед, ничьих и поражений.
9. Сохранение статистики в файл. После обновления статистики программа автоматически сохраняет всю таблицу рекордов в текстовый файл «records.txt». Если файл не существует, он создаётся; если существует — перезаписывается.
10. Загрузка статистики из файла. При запуске программа проверяет наличие файла «records.txt». Если файл существует и соответствует формату (имя,

победы, ничьи, поражения), данные загружаются в вектор записей. Если файл не существует, программа начинает с пустой таблицей рекордов.

11. Отображение игрового поля. На протяжении всей игры после каждого хода программа очищает экран и выводит текущее состояние поля в псевдографическом виде с нумерацией строк и столбцов, а также с разделителями в виде символов '|' и '-' для наглядности.

Описание разработки программы

Для написания данной программы использован язык программирования C++ и MASM (ассемблерные вставки). В качестве среды программирования был выбран программный продукт Microsoft Visual Studio 2022, который обладает всеми необходимыми средствами для разработки и отладки приложений, включая поддержку встроенного ассемблера.

Проверка победных комбинаций в разработанной программе реализуется на языке Assembler с использованием вставки `__asm`, что позволяет оптимизировать проверку всех восьми возможных линий (три строки, три столбца, две диагонали) путём прямого доступа к ячейкам памяти.

Сохранение информации реализуется в текстовый файл «records.txt», который создаётся автоматически при первой игре и обновляется после каждой партии. В файле хранится статистика всех игроков: имя, количество побед, количество ничьих, количество поражений.

Интерфейс информационной системы представлен в виде последовательного игрового процесса с элементами меню (запрос имени, игровое поле, запрос на повтор игры), т.к. он является наиболее удобным для консольного приложения. Управление программой будет осуществляться с помощью клавиатуры: ввод имени игрока (строка латиницей), ввод координат хода (два числа от 1 до 3), выбор продолжения игры (y/n).

Разработка программы началась с реализации набора функций, необходимых для работы с игрой «Крестики-нолики»:

- Для инициализации игрового поля (заполнение всех клеток пробелами) используется функция `initializeBoard()`.
- Для отображения текущего состояния игрового поля на экране в псевдографическом формате (с использованием символов |, -, нумерацией строк и столбцов) используется функция `printBoard()`.
- Для очистки экрана консоли используется функция `clearScreen()`.
- Для очистки экрана консоли используется функция `clearScreen()`.

- Для проверки заполненности игрового поля (условие ничьей) используется функция `isBoardFull()`.
- Для проверки корректности вводимого хода (клетка существует и свободна) используется функция `isValidMove(int row, int col)`.
- Для оценки текущего состояния поля с точки зрения алгоритма минимакса используется функция `evaluateBoard()`.
- Для реализации искусственного интеллекта компьютера с использованием рекурсивного алгоритма минимакса используется функция `minimax()`.
- Для выполнения хода компьютером (выбор оптимальной клетки на основе оценки минимакса) используется функция `computerMove()`.
- Для загрузки статистики игроков из файла «records.txt» используется функция `loadRecords()`.
- Для сохранения статистики игроков в файл «records.txt» используется функция `saveRecords()`.
- Для поиска или создания записи об игроке в векторе статистики используется функция `findOrCreateRecord()`.
- Для отображения таблицы рекордов (сортировка игроков по количеству побед) используется функция `showLeaderboard()`.
- Для запуска основного игрового цикла используется функция `startGame()`, которая объединяет весь игровой процесс.
- Для главной точки входа в программу используется функция `main()`, в которой происходит настройка кодировки, вывод заставки и вызов `startGame()`.
- Особенностью данной разработки является использование ассемблерной вставки в функции `checkWinner()`, которая выполняется быстрее аналогичной реализации на C++ за счёт низкоуровневого доступа к памяти и минимизации накладных расходов на вызовы функций. Ассемблерный код последовательно сравнивает содержимое ячеек игрового поля с заданным символом, проверяя все возможные победные комбинации, и при обнаружении полной линии немедленно возвращает результат.

Отладка и тестирование

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Программа обладает всеми средствами необходимыми при разработке и отладке программы. Для отладки использовались несколько возможностей Visual Studio: точка останова, трассировка, анализ содержимого переменных.

В результате проведения тестирования были выявлены и устранены все ошибки, программа работает стабильно, корректно обрабатывает все игровые ситуации, правильно определяет победу и ничью, сохраняет и загружает статистику игроков. Ассемблерная вставка успешно выполняет проверку победных комбинаций, что подтверждено в ходе отладки с анализом содержимого памяти и регистров процессора.

Описание программы

Приложение «TicTacToe.exe» является основным модулем программы. При запуске программы происходит вывод заставки с информацией о курсовой работе, после чего программа запрашивает имя игрока и готова к началу игрового процесса (таблица 1). Затем, вводя координаты ходов, игрок взаимодействует с программой, которая после каждого хода отображает текущее состояние игрового поля и проверяет условия победы или ничьей. Выход из программы осуществляется после завершения партии при ответе «n» на вопрос «Сыграем ещё?» или сразу после запуска при желании пользователя.

Таблица 1 — Описание всех функциональных модулей

Модуль	Функции	Назначение
Главный модуль	main()	Установка кодировки, вывод заставки, запуск игрового процесса
Модуль игровой логики	startGame() initializeBoard() printBoard() clearScreen() isValidMove() isBoardFull()	Управление игровым процессом, отображение поля, проверка корректности ходов
Модуль проверки победы	checkWinner()	Проверка победных комбинаций на ассемблере
Модуль искусственного интеллекта	evaluateBoard() minimax() computerMove()	Реализация хода компьютера с использованием алгоритма минимакса

Продолжение таблицы 1

Модуль работы со статистикой	loadRecords() saveRecords() findOrCreateRecord() showLeaderboard()	Загрузка, сохранение и отображение статистики игроков
------------------------------	---	---

Для запуска основного игрового процесса, управления игровым циклом, обработки ходов игрока и компьютера, а также для ведения статистики используется функция `startGame()` (рисунок 2).

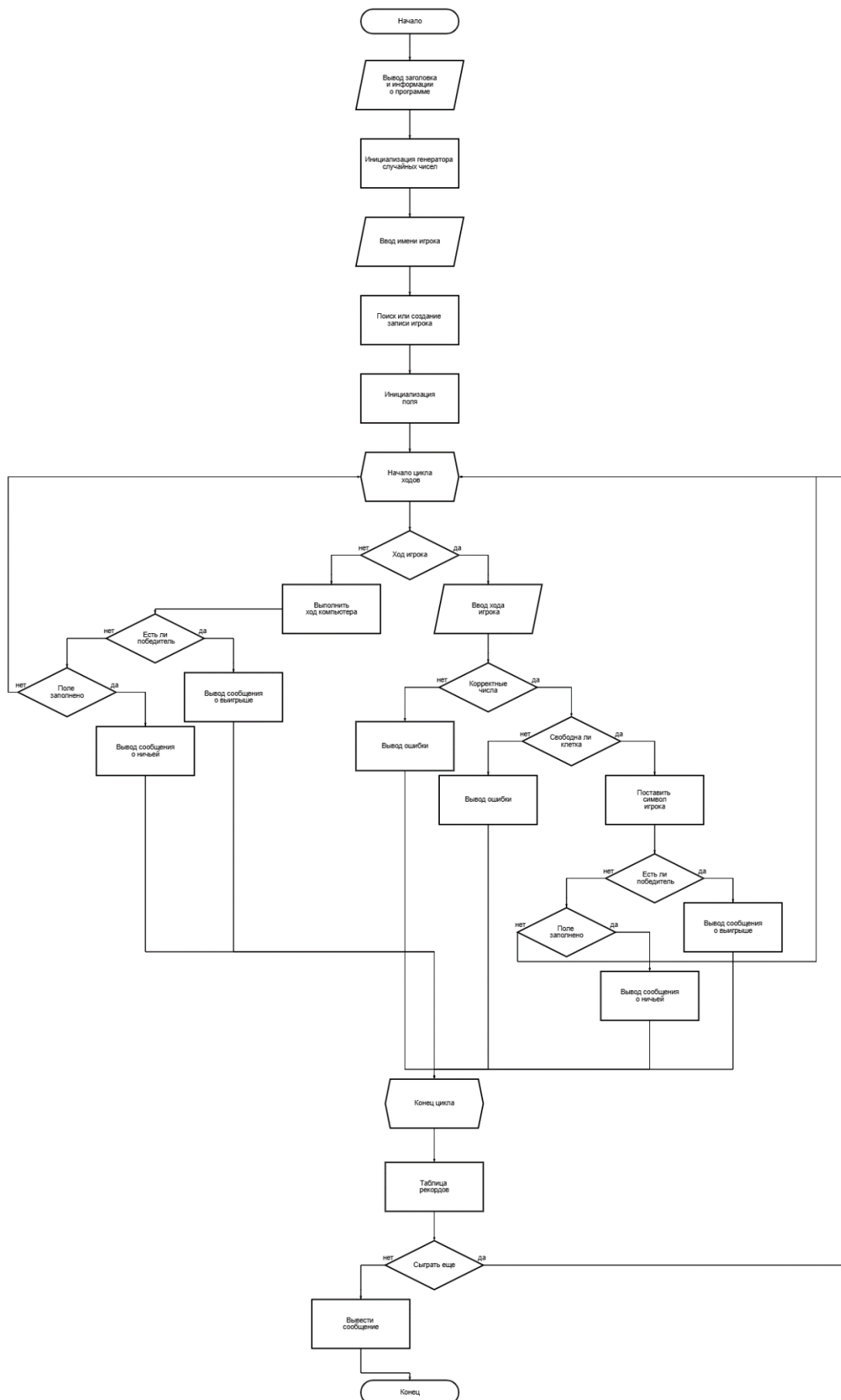


Рисунок 2 – Блок схема функции startGame

Для инициализации игрового поля, очистки всех клеток перед началом новой партии используется функция `initializeBoard()` (рисунок 3).

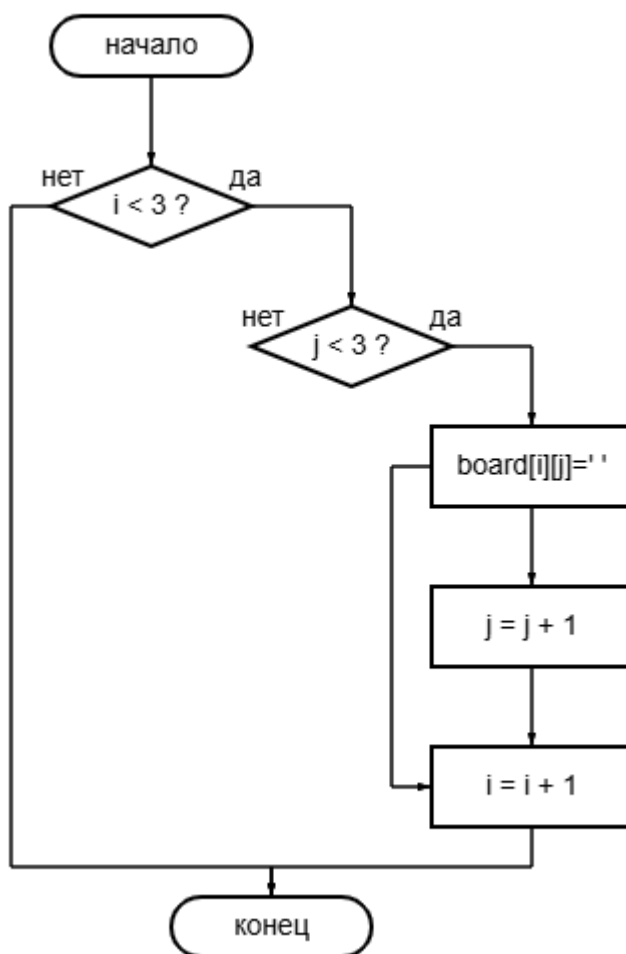


Рисунок 3 – Блок схема функции `initializeBoard`

Функция `printBoard()` предназначена для отображения текущего состояния игрового поля в консоли в псевдографическом виде. Она выводит нумерацию строк и столбцов, содержимое каждой клетки (крестик, нолик или пробел), а также разделительные линии для наглядного представления поля 3×3 .

Функция `isValidMove()` предназначена для проверки корректности хода игрока. Она определяет, находится ли введённая клетка в пределах игрового поля (от 1 до 3 по строке и столбцу) и свободна ли она (не занята символом 'X' или 'O').

Функция `isBoardFull()` предназначена для проверки заполненности игрового поля. Она определяет, остались ли на поле свободные клетки (пробелы). Если все девять клеток заняты символами 'X' или 'O', функция возвращает `true`,

что означает наступление ничьей. Если хотя бы одна клетка пуста, функция возвращает false, и игра продолжается.

Функция checkWinner() предназначена для проверки наличия победной комбинации на игровом поле для заданного символа ('X' или 'O'). Проверка осуществляется с использованием ассемблерной вставки, которая последовательно анализирует все восемь возможных победных линий: три строки, три столбца и две диагонали (рисунок 4).

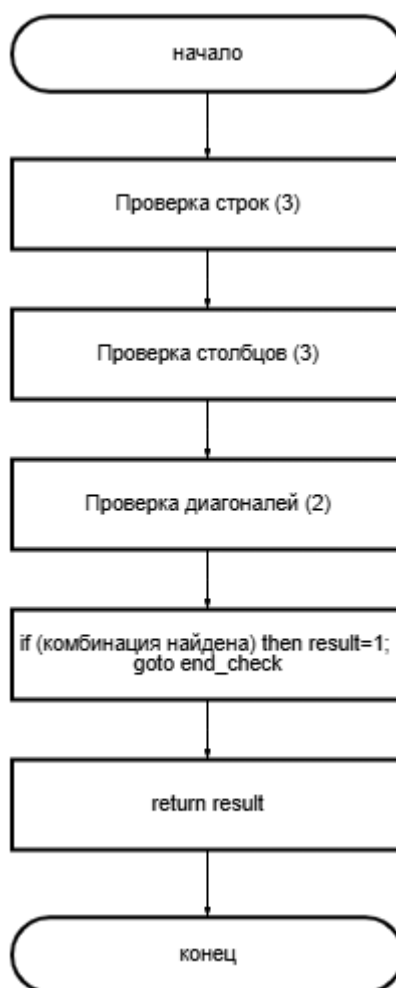


Рисунок 4 – Блок схема функции checkWinner

Функция evaluateBoard() предназначена для оценки текущего состояния игрового поля с точки зрения алгоритма минимакса (minimax). Она возвращает числовое значение, которое показывает, насколько выгодна текущая позиция для компьютера.

Функция minimax() реализует рекурсивный алгоритм минимакса, который используется компьютером для выбора оптимального хода. Алгоритм перебирает

все возможные варианты развития игры и оценивает их, выбирая наилучший ход для компьютера и наихудший для игрока (рисунок 5).

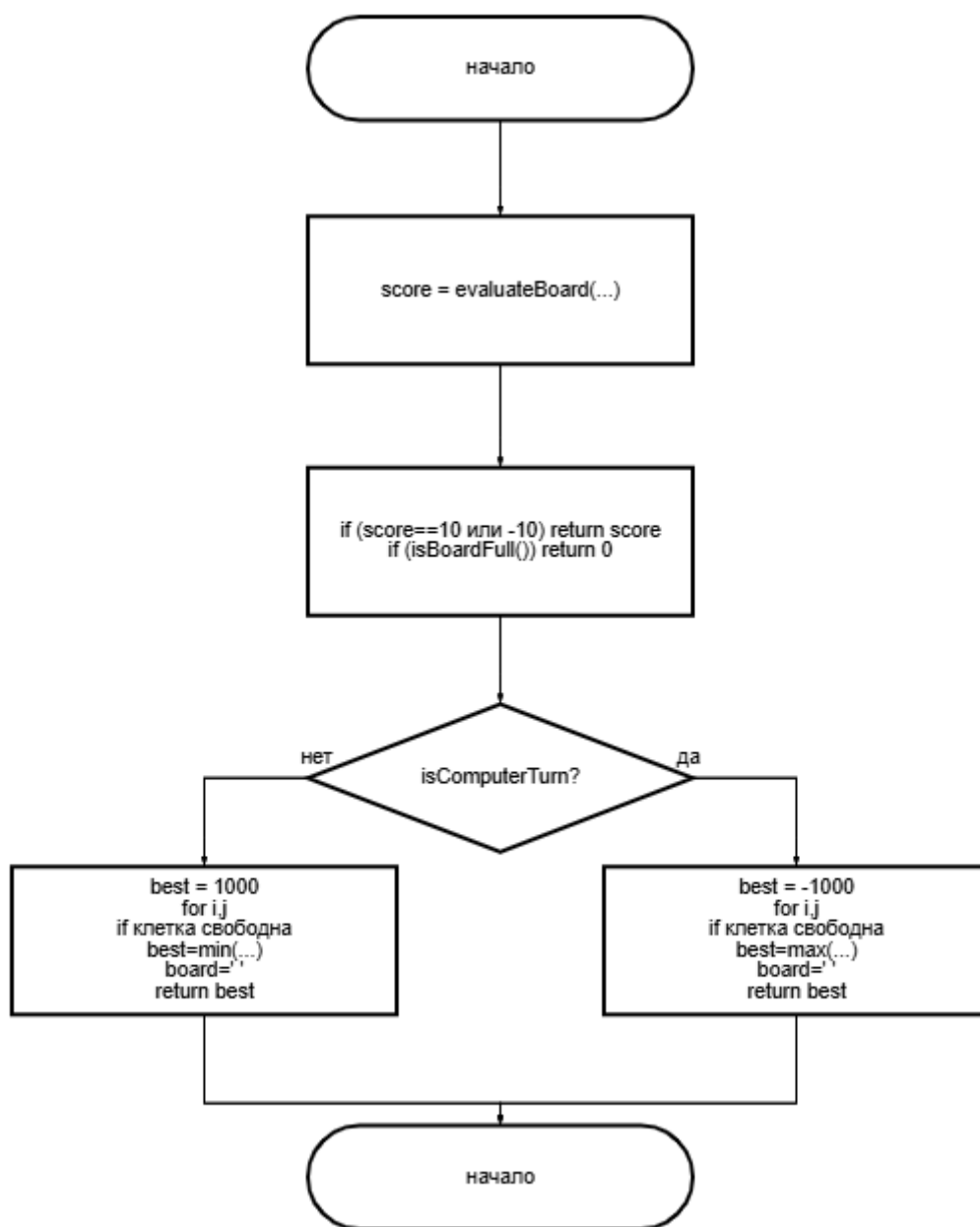


Рисунок 5 – Блок схема функции minimax

Функция `computerMove()` перебирает все свободные клетки, временно делает в каждой ход компьютера, вызывает `minimax` для оценки получившейся позиции, и выбирает клетку с максимальной оценкой.

Функция `loadRecords()` предназначена для загрузки статистики игроков из текстового файла «records.txt».

Функция `saveRecords()` записывает все данные из вектора `records` (имя игрока, количество побед, ничьих и поражений) в файл для долговременного хранения.

Функция `findOrCreateRecord()` предназначена для поиска записи об игроке в векторе статистики. Если запись с указанным именем существует, функция возвращает ссылку на неё. Если запись не найдена, функция создаёт новую запись с нулевыми показателями, добавляет её в вектор и возвращает ссылку на созданную запись.

Функция `showLeaderboard()` предназначена для отображения таблицы рекордов на экране. Она сортирует записи игроков по количеству побед (от большего к меньшему) и выводит их в виде форматированной таблицы, содержащей имя игрока, количество побед, ничьих и поражений.

Интеграция всех представленных функций в единую систему обеспечивает корректное выполнение алгоритмов игры, сохранение и загрузку статистики, а также удобное взаимодействие с пользователем, что в полной мере соответствует предъявленным требованиям.

Руководство пользователя

Программа «TicTacToe.exe» предназначена для игры в «Крестики-нолики» между пользователем и компьютером. Программа имеет интуитивно понятный интерфейс, реализованный в консольном режиме с использованием псевдографики, и поддерживает такие операции как: выход из программы, начало новой игры, ввод имени игрока, выполнение хода игроком, автоматический ход компьютера с использованием алгоритма минимакса, проверку победных комбинаций с помощью ассемблерной вставки, определение ничьей, отображение текущего состояния игрового поля, сохранение статистики игроков в файл, загрузку статистики из файла, отображение таблицы рекордов, отсортированной по количеству побед, а также повторную игру после завершения партии.

При запуске программы пользователю отображается заставка с информацией о курсовой работе и сообщение: «Введите ваше имя:». Пользователь должен ввести своё имя, используя латинские буквы (рисунок 15).

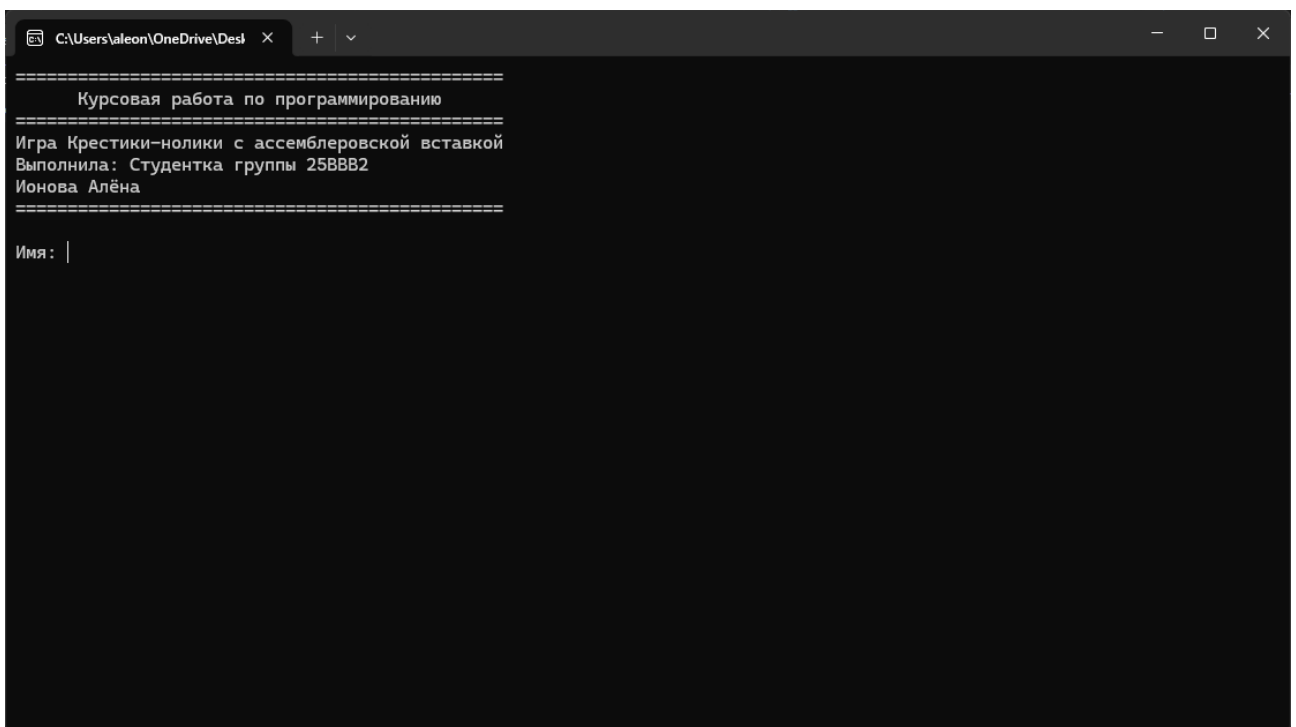


Рисунок 15 – Регистрация игрока

После того как пользователь ввёл своё имя, программа переходит к основному игровому процессу (рисунок 16).

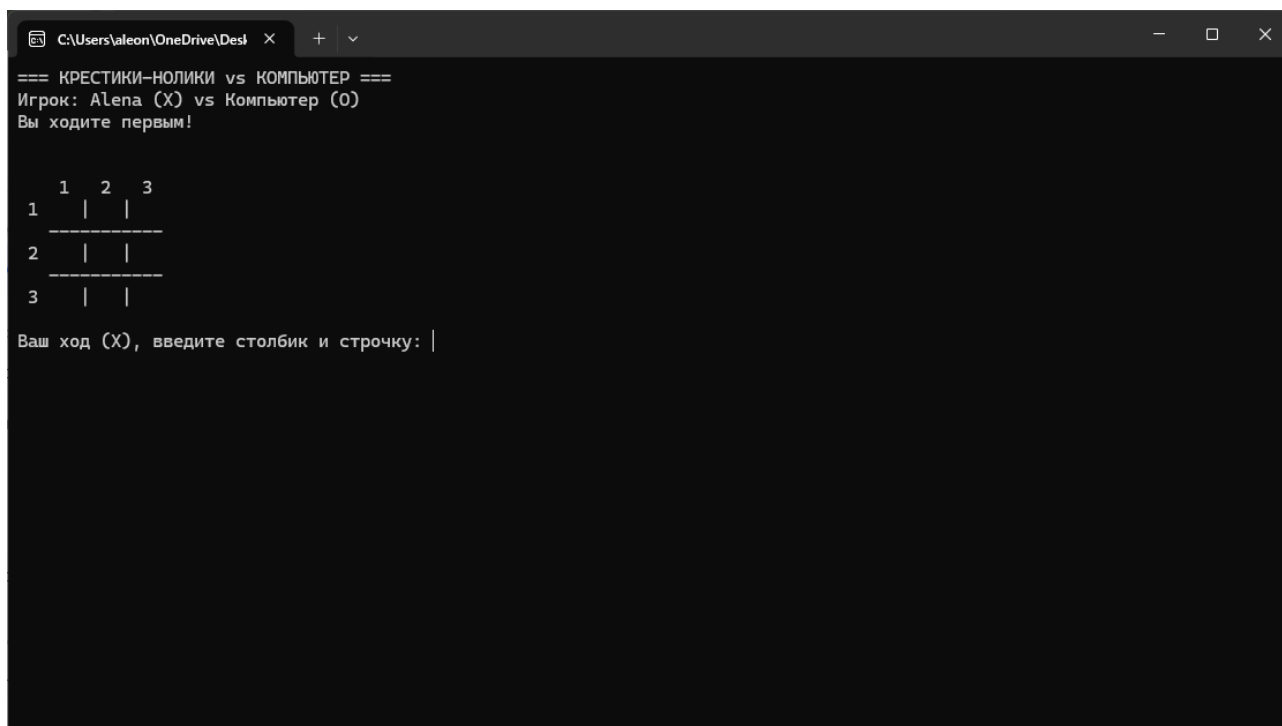


Рисунок 16 – Выход из программы

Программа информирует пользователя об исходе партии, выводя на экран одно из сообщений: победа игрока, победа компьютера или ничья. Выводит таблицу рекордов и сообщение: «Сыграем еще? (y/n)» Программа ожидает ввод символа с клавиатуры (рисунок 17).

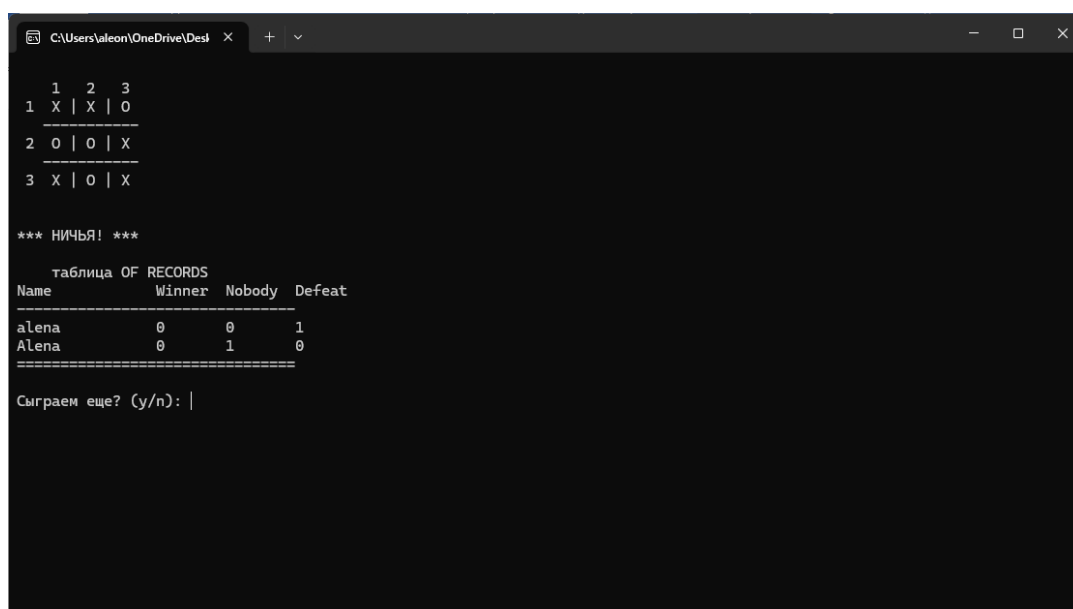


Рисунок 17 – Результаты игры и таблица рекордов

Руководство наглядно демонстрирует все возможности программы, позволяя пользователю быстро разобраться в её функционале.

Заключение

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ. Была освоена работа с файлами (сохранение и загрузка статистики игроков в файл «records.txt»), изучены функции работы с консолью (очистка экрана, установка кодировки для отображения русского языка, псевдографический вывод игрового поля) и оперативной памятью (использование векторов для динамического хранения записей статистики). Также были получены основные навыки отладки и тестирования программ (использование точек останова, трассировка, анализ содержимого переменных и регистров при выполнении ассемблерных вставок) и программирования в среде Microsoft Visual Studio 2022 на языках Си/C++ и Assembler (с использованием встроенных ассемблерных вставок `__asm` для оптимизации проверки победных комбинаций).

В рамках курсовой работы была написана программа для игры в «Крестики-нолики» между пользователем и компьютером. Она имеет все необходимые функции для полноценного игрового процесса: инициализацию и отображение игрового поля, проверку победы (с ассемблерной вставкой), проверку ничьей, ход компьютера на основе алгоритма минимакса, сохранение и загрузку статистики игроков, отображение таблицы рекордов, а также возможность повторной игры.

В дальнейшем программу можно улучшить путём добавления графического интерфейса, что позволит улучшить визуальное восприятие игры и навигацию по меню, а также путём добавления возможности использования мыши для выбора клеток игрового поля. Кроме того, можно реализовать режим игры «человек против человека», возможность выбора размера игрового поля (4×4, 5×5 и более с изменением правила победы), добавление звукового сопровождения и расширенную статистику с графиками побед.

Список литературы

1. Павловская Т.А. С++. Программирование на языке высокого уровня. – СПб.: Питер, 2019. – 464 с.
2. Абель П. Язык ассемблера для IBM PC и программирования. – М.: Вильямс, 2017. – 736 с.
3. Рассел С., Норвиг П. Искусственный интеллект. Современный подход. – М.: Вильямс, 2021. – 1408 с.
4. Собкин М.С. Assembler. Самоучитель. – М.: КУДИЦ-Образ, 2018. – 368 с.
5. Калашников О. А. Ассемблер – это просто. Учимся программировать / О. А. Калашников. — Санкт-Петербург : БХВ-Петербург, 2019. — 320 с. — ISBN 978-5-9775-6543-7.
6. Юров В. И. Assembler. Практикум / В. И. Юров. — Санкт-Петербург : Питер, 2018. — 400 с. — ISBN 978-5-496-00918-2.

Приложение А
Листинги программы
game.cpp

```
#include "game.h"
#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>
#include <vector>
#include <algorithm>
#include <string>
#include <windows.h>

using namespace std;

char board[3][3];

struct Record {
    string playerName;
    int wins;
    int draws;
    int losses;
};

vector<Record> records;

void initializeBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
```

```

        board[i][j] = ' ';
    }
}

void clearScreen() {
    system("cls");
}

void printBoard() {
    cout << "\n";
    cout << "    1    2    3\n";

    for (int i = 0; i < 3; i++) {
        cout << " " << i + 1 << " ";

        for (int j = 0; j < 3; j++) {
            cout << " " << board[i][j] << " ";
            if (j < 2) {
                cout << "|";
            }
        }

        cout << "\n";

        if (i < 2) {
            cout << "    -----\n";
        }
    }

    cout << "\n";

```

```
}
```

```
bool checkWinner(char symbol) {
```

```
    bool result = false;
```

```
    __asm {
```

```
        mov al, symbol
```

```
        mov bl, al
```

```
        mov al, board[0]
```

```
        cmp al, bl
```

```
        jne row1_check
```

```
        mov al, board[1]
```

```
        cmp al, bl
```

```
        jne row1_check
```

```
        mov al, board[2]
```

```
        cmp al, bl
```

```
        jne row1_check
```

```
        mov result, 1
```

```
        jmp end_check
```

```
    row1_check :
```

```
    mov al, board[3]
```

```
        cmp al, bl
```

```
        jne row2_check
```

```
        mov al, board[4]
```

```
        cmp al, bl
```

```
        jne row2_check
```

```
        mov al, board[5]
```

```
        cmp al, bl
```

```
        jne row2_check
```

```

        mov result, 1
        jmp end_check

row2_check :
mov al, board[6]
    cmp al, bl
    jne col0_check
    mov al, board[7]
    cmp al, bl
    jne col0_check
    mov al, board[8]
    cmp al, bl
    jne col0_check
    mov result, 1
    jmp end_check

col0_check :
mov al, board[0]
    cmp al, bl
    jne col1_check
    mov al, board[3]
    cmp al, bl
    jne col1_check
    mov al, board[6]
    cmp al, bl
    jne col1_check
    mov result, 1
    jmp end_check

col1_check :
mov al, board[1]

```

```

        cmp al, bl
        jne col2_check
        mov al, board[4]
        cmp al, bl
        jne col2_check
        mov al, board[7]
        cmp al, bl
        jne col2_check
        mov result, 1
        jmp end_check

col2_check :
mov al, board[2]
        cmp al, bl
        jne diag1_check
        mov al, board[5]
        cmp al, bl
        jne diag1_check
        mov al, board[8]
        cmp al, bl
        jne diag1_check
        mov result, 1
        jmp end_check

diag1_check :
mov al, board[0]
        cmp al, bl
        jne diag2_check
        mov al, board[4]
        cmp al, bl
        jne diag2_check

```

```

        mov al, board[8]
        cmp al, bl
        jne diag2_check
        mov result, 1
        jmp end_check

diag2_check :
mov al, board[2]
    cmp al, bl
    jne end_check
    mov al, board[4]
    cmp al, bl
    jne end_check
    mov al, board[6]
    cmp al, bl
    jne end_check
    mov result, 1

    end_check:
}

return result;
}

bool isBoardFull() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false;
            }
        }
    }
}

```

```

    }
    return true;
}

bool isValidMove(int row, int col) {
    return row >= 1 && row <= 3 &&
           col >= 1 && col <= 3 &&
           board[row - 1][col - 1] == ' ';
}

int evaluateBoard(char computerSymbol, char playerSymbol)
{
    if (checkWinner(computerSymbol)) return 10;
    if (checkWinner(playerSymbol)) return -10;
    return 0;
}

int minimax(bool isComputerTurn, char computerSymbol, char
playerSymbol, int depth) {
    int score = evaluateBoard(computerSymbol,
playerSymbol);

    if (score == 10 || score == -10) return score;
    if (isBoardFull()) return 0;

    if (isComputerTurn) {
        int best = -1000;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    board[i][j] = computerSymbol;

```

```

        best    =    max(best,    minimax(false,
computerSymbol, playerSymbol, depth + 1));
        board[i][j] = ' ';
    }
}
}
return best;
}
else {
    int best = 1000;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                board[i][j] = playerSymbol;
                best    =    min(best,    minimax(true,
computerSymbol, playerSymbol, depth + 1));
                board[i][j] = ' ';
            }
        }
    }
    return best;
}
}

```

```

void computerMove(char computerSymbol, char playerSymbol)
{
    int bestScore = -1000;
    int bestRow = -1, bestCol = -1;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {

```

```

        if (board[i][j] == ' ') {
            board[i][j] = computerSymbol;
            int moveScore = minimax(false,
computerSymbol, playerSymbol, 0);
            board[i][j] = ' ';

            if (moveScore > bestScore) {
                bestScore = moveScore;
                bestRow = i;
                bestCol = j;
            }
        }
    }
}

if (bestRow != -1 && bestCol != -1) {
    board[bestRow][bestCol] = computerSymbol;
}
}

void loadRecords() {
    ifstream file("records.txt");
    if (file.is_open()) {
        records.clear();
        Record r;
        while (file >> r.playerName >> r.wins >> r.draws
>> r.losses) {
            records.push_back(r);
        }
        file.close();
    }
}

```

```

}

void saveRecords() {
    ofstream file("records.txt");
    if (file.is_open()) {
        for (const auto& r : records) {
            file << r.playerName << " " << r.wins << " "
<< r.draws << " " << r.losses << "\n";
        }
        file.close();
    }
}

```

```

Record& findOrCreateRecord(const string& playerName) {
    for (auto& r : records) {
        if (r.playerName == playerName) {
            return r;
        }
    }
    Record newRecord;
    newRecord.playerName = playerName;
    newRecord.wins = 0;
    newRecord.draws = 0;
    newRecord.losses = 0;
    records.push_back(newRecord);
    return records.back();
}

```

```

void showLeaderboard() {
    cout << "\n    таблица OF RECORDS    \n";
    cout << "Name\t\tWinner\tNobody\tDefeat\n";
}

```

```

    cout << "-----\n";

    sort(records.begin(), records.end(), [](const Record&
a, const Record& b) {
        return a.wins > b.wins;
    });

    for (const auto& r : records) {
        cout << r.playerName << "\t\t" << r.wins << "\t"
<< r.draws << "\t" << r.losses << "\n";
    }
    cout << "=====\n\n";
}

void startGame() {
    cout << "=====" << endl;
    cout << "          Курсовая работа по программированию          "
<< endl;
    cout << "=====" << endl;
    cout << "Игра Крестики-нолики с ассемблеровской
вставкой " << endl;
    cout << "Выполнила: Студентка группы 25ВВВ2" << endl;
    cout << "Ионова Алёна" << endl;
    cout << "=====" << endl;
    cout << endl;
    cout << endl;

```

```

srand(time(0));
loadRecords();

string playerName;
cout << "Введите ваше имя: ";
cin >> playerName;

Record& playerRecord = findOrCreateRecord(playerName);

char playAgain;
do {
    initializeBoard();

    char playerSymbol = 'X';
    char computerSymbol = 'O';
    bool playerTurn = true;

    while (true) {
        clearScreen();
        cout << "=== КРЕСТИКИ-НОЛИКИ vs КОМПЬЮТЕР
===\n";

        cout << "Игрок: " << playerName << " (" <<
playerSymbol << ") vs Компьютер (" << computerSymbol <<
")\n";

        cout << "Вы ходите первым!\n\n";
        printBoard();

        if (playerTurn) {
            int row, col;
            cout << "Ваш ход (" << playerSymbol << "),
введите столбик и строчку: ";

```

```

while (!(cin >> row >> col)) {
    cin.clear();
    cin.ignore(1000, '\n');
    cout << "Неверный ввод. Введите два
числа: ";
}

if (!isValidMove(row, col)) {
    cout << "Неверный ход! Нажмите Enter
чтобы продолжить...";
    cin.ignore(1000, '\n');
    cin.get();
    continue;
}

board[row - 1][col - 1] = playerSymbol;

if (checkWinner(playerSymbol)) {
    clearScreen();
    printBoard();
    cout << "\n*** ВЫ ПОБЕДИЛИ! ***\n";
    playerRecord.wins++;
    saveRecords();
    break;
}

if (isBoardFull()) {
    clearScreen();
    printBoard();
    cout << "\n*** НИЧЬЯ! ***\n";

```

```

        playerRecord.draws++;
        saveRecords();
        break;
    }

    playerTurn = false;
}
else {
    cout << "Компьютер думает...\n";
    computerMove(computerSymbol,
playerSymbol);

    if (checkWinner(computerSymbol)) {
        clearScreen();
        printBoard();
        cout << "\n*** КОМПЬЮТЕР ПОБЕДИЛ!
***\n";

        playerRecord.losses++;
        saveRecords();
        break;
    }

    if (isBoardFull()) {
        clearScreen();
        printBoard();
        cout << "\n*** НИЧЬЯ! ***\n";
        playerRecord.draws++;
        saveRecords();
        break;
    }
}

```

```

        playerTurn = true;
    }
}

showLeaderboard();

cout << "Сыграем еще? (y/n): ";
cin >> playAgain;

} while (playAgain == 'y' || playAgain == 'Y');

cout << "\nСпасибо за игру! Ваша статистика
сохранена.\n";
}

```

mainn.cpp

```
#include <windows.h>
#include "game.h"
```

```
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    startGame();
    return 0;
}
```

game.h

```
#ifndef GAME_H
#define GAME_H

void startGame();

#endif
```