Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт кибербезопасности и защиты информации

КУРСОВАЯ РАБОТА

Игра «Крестики-нолики на бесконечном поле»

по дисциплине «Структуры данных»

Выполнил
студент гр. 4851003/20002 Анисимов Е.И.

<подпись>

Преподаватель
асс. преподавателя Панков И.Д.

<подпись>

«___» _____ 2023 г.

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Темой для курсовой работы была выбрана реализация игры “Крестики-нолики на бесконечном поле” на языке программирования Си. Для реализации интерфейса игры использовался фреймворк OpenGL. Он позволяет нам создавать графику для игры и управлять всеми элементами интерфейса. Для создания игры мы использовали различные алгоритмы и структуры данных.

1 Обзор задач и правил игры

Крестики-нолики – логическая игра между двумя противниками на квадратном поле любого размера вплоть до бесконечного. Один из игроков играет “крестиками”, а второй – “ноликами”. Игроки по очереди ставят на свободные клетки поля свой знак, за который он играет.

Цель игры состоит в том, чтобы выставить на поле определенное количество знаков в ряд. Тот, кто первым достигает этой цели – выигрывает, а второй игрок автоматически проигрывает.

Клетки поля могут быть в трех состояниях: нолик, крестик и пустая клетка. В зависимости от хода конкретного игрока, в них может появляться крестик или нолик.

1.1 Правила игры

Для того чтобы начать игру, пользователь должен перейти в настройки и задать определенные параметры для игры. В настройках можно изменить размер поля, длину выигрышной линии, сложность игры и определить знак, за который играет пользователь.

В игре есть четыре уровня сложности: 1 – easy, 2 – medium, 3 – hard, 4 – very hard. Для каждого уровня используется определенный алгоритм для робота.

Во время игры пользователь в любой момент может выйти из игры, при этом его прогресс сохраняется. Если на следующем запуске программы пользователь кликнет на кнопку “Load”, то игра, которую он не завершил в прошлый раз возобновится с того момента, на котором она остановилась.

1.2 Задачи

Необходимо реализовать игру с учетом всех правил, описанных выше в пункте 1.1. Кроме того, необходимо выполнить следующие условия:

1. Реализация игры на языке Си.
2. Создание четырех уровней сложности.

3. Наличие в игре главного меню, раздела «О программе», справки с информацией, таблицы рекордов (запись в нее добавляется по окончанию игры, если игрок выиграл, у игрока спрашивают имя), выбора уровня сложности.

4. Хранение данных во внешних файлах, размещенных в каталогах программы.

2 Теоретические исследования

Далее будут описаны некоторые особенности реализации игры “Крестики-нолики на бесконечном поле”.

2.1 Игровое поле

Основной структурой, описывающей игровые события является массив структур Cell **field, размер которого задается переменной FIELD_SIZE, задаваемой пользователем. Структура Cell содержит в себе описание состояния клетки state и две вещественные координаты, которые используются для отрисовки фигур (это центры каждого квадрата на поле в масштабе пикселей).

```
typedef struct{
    float c_x;
    float c_y;
    enum state_ceil state;
}Cell;
```

Рисунок 1 – Структура для описания поля

Состояние клетки описывается перечислением enum state_ceil, которое может принимать значения FREE, CROSS, CIRCLE.

```
enum state_ceil{
    FREE = 0,
    CROSS = 1,
    CIRCLE = 2
};
```

Рисунок 2 – Перечисление для описания состояния клетки.

Кроме основного массива, описывающего состояние игрового поля, имеется массив, описывающий состояние видимого поля Cell **view_field. Каждый элемент в нем аналогично представляет собой структуру Cell. Размер этого массива задается переменной VIEW_SIZE, которая вычисляется в функции init() по следующему правилу: если размер игрового поля, введенного пользователем, не более 10 клеток, то VIEW_SIZE равен FIELD_SIZE, в противном случае его размер равен константе MAX_VIEW_SIZE, равной 10, т.е. максимальный размер отображаемого поля не превышает 10 клеток.

В случае, когда размер игрового поля превышает размер отображаемого, перемещение по игровому полю осуществляется с помощью клавиш стрелок.

2.2 Структуры данных

Кроме вышеописанных структур, констант и переменных, программа имеет следующие структуры, перечисления, глобальные переменные и константы:

```
#define MIN_SIZE_FIELD 3
#define MAX_SIZE_FIELD 10000
#define MAX_VIEW_SIZE 10
#define WINDOW_SIZE 700
```

Рисунок 3 – Использованные константы

MIN_SIZE_FIELD – определяет минимальный размер игрового поля, MAX_SIZE_FIELD – определяет максимальный размер игрового поля, WINDOW_SIZE – определяет размер окна в пикселях.

```
typedef struct{
    int x;
    int y;
}move;
enum move{
    PLAYER,
    AI
};
enum state_ceil{
    FREE = 0,
    CROSS = 1,
    CIRCLE = 2
};
enum state_game{
    GAME_CONT = 0,
    WIN_CROSS = 1,
    WIN_CIRCLE = 2,
    DRAW = 3
};
typedef struct{
    float c_x;
    float c_y;
    enum state_ceil state;
}Cell;
```

Рисунок 4 – Использованные перечисления и структуры

struct move – структура, определяющая координаты для хода в масштабе клеток, enum move – перечисление, определяет чей текущий ход, enum state_game – перечисление, определяющее состояние игры после каждого хода.

```
clock_t t1, t2 = 0;

int WIN_LEN;
int FIELD_SIZE;
int VIEW_SIZE;

char fieldSize[10000];
char winLen[10000];
char diffLevel[10];
char whoStart[10];
char userName[100];

int flagStart = 0;
int flagSettings = 0;
int flagAbout = 0;
int flagWin = 0;
int flagPause = 1;
int flagRecords = 0;
int playerWin = 0;
int settingsChoise = 0;
int current_x = 0;
int current_y = 0;
int LVL;
int choice;

enum state_game state_game = GAME_CONT;
enum state_ceil current_move;
enum state_ceil ai_move;
enum state_ceil player_move;
unsigned int amount_moves;
unsigned int free_cell;

Cell **field;
Cell **view_field;
```

Рисунок 5 – Использованные глобальные переменные

WIN_LEN – длина линии для выигрыша, LVL – уровень сложности игры компьютера, choice – выбор пользователем фигуры для игры (1 – крестики, 2 – нолики), current_x – координата по x левой нижней клетки видимого поля относительно игрового поля, current_y – координата по y левой нижней клетки видимого поля относительно игрового поля, state_game – текущее состояние игры, current move – определяет, чей ход текущий, ai_move – фигура для игры компьютером, player_move – фигура для игры пользователя, amount_moves – количество совершенных ходов, free_cell – количество оставшихся свободных клеток.

2.3 OpenGL

OpenGL (Open Graphics Library) – это кроссплатформенный, открытый стандарт графической библиотеки состоящей из набора функций, которые позволяют программистам создавать трехмерную и двухмерную графику для интерактивных приложений. Он используется в различных областях, таких как игровая индустрия, визуализация данных, научные вычисления и другие приложения, где требуется отображение графики. OpenGL позволяет работать с трехмерной графикой независимо от аппаратной платформы, что делает его популярным среди разработчиков программного обеспечения.

OpenGL может выполнять множество графических операций, некоторые из которых включают в себя:

1. Режимы вывода графики, такие как режим отрисовки точек, линий и треугольников.
2. Матричные преобразования, такие как перемещение, масштабирование и поворот объектов.
3. Работа с текстурами, включая чтение изображений, создание текстур и наложение их на объекты.
4. Затенение объектов, включая различные методы затенения, такие как режимы освещения и теней.
5. Работа с буферами изображений, включая создание и управление буферами кадров и глубины.
6. Управление устройствами вывода графики и настройка их параметров, таких как разрешение, частота кадров и т.д.

На практике, почти все графические приложения, использующие OpenGL, выполняют некоторые комбинации этих операций, чтобы создать желаемый визуальный эффект.

3 Графика и реализация

3.1 Функционал приложения

В разработанном приложении имеется меню:

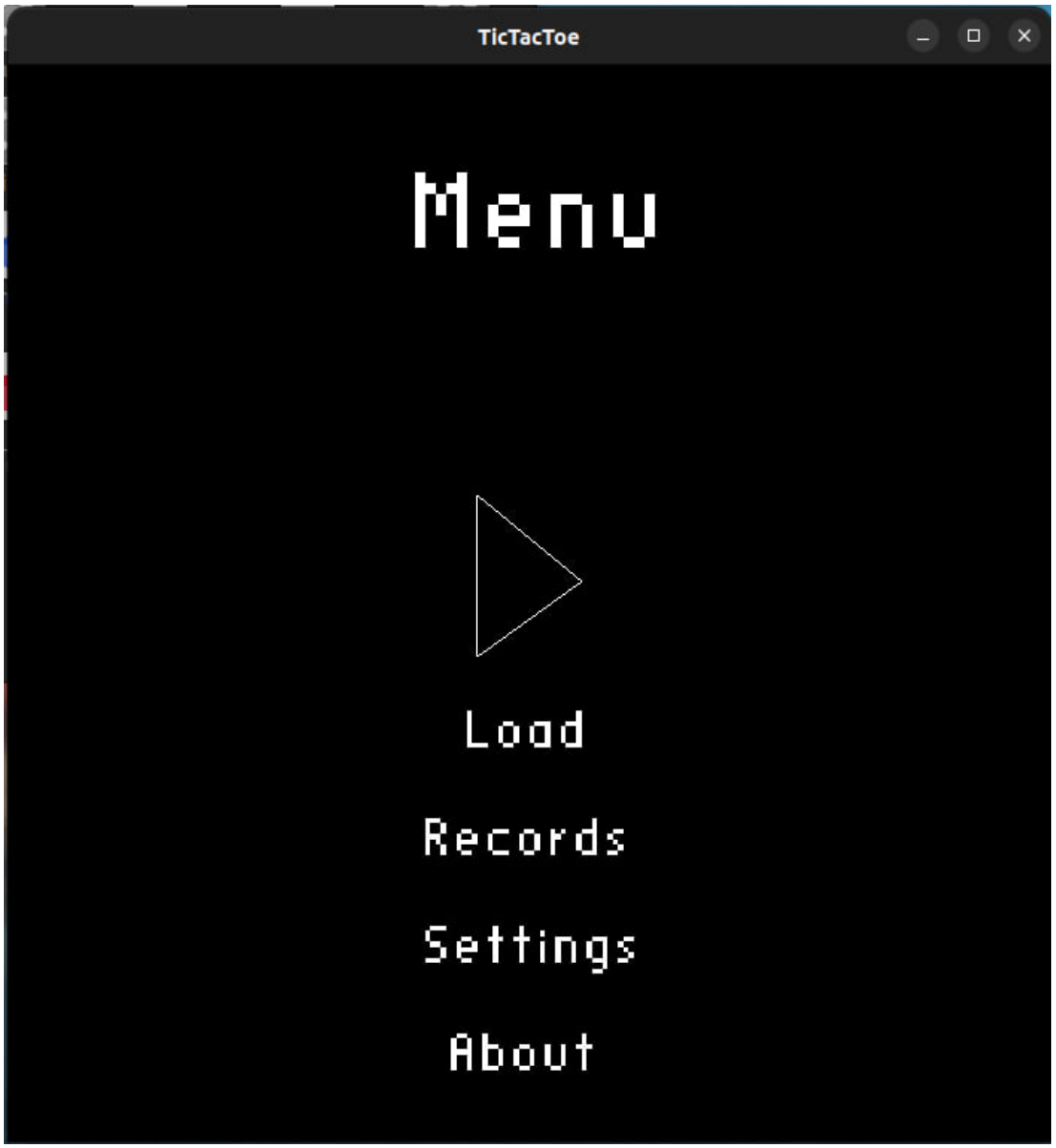


Рисунок 6 – Экран Menu

Если пользователь нажимает на кнопку “Settings”, то открывается окно настроек, в котором можно задать размер поля, длину выигрышной линии, сложность алгоритма и тот знак, за который пользователь хочет играть.

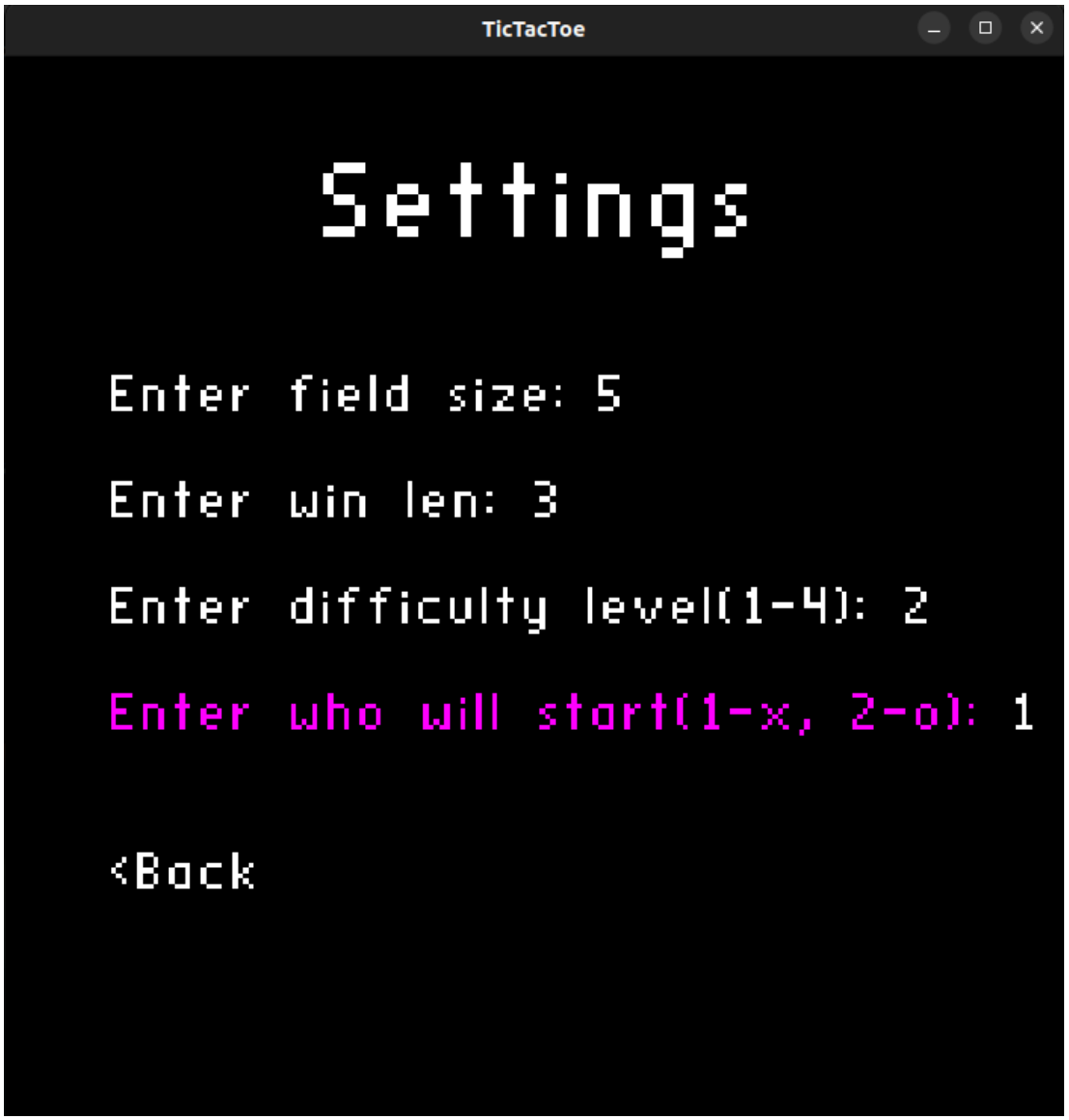


Рисунок 7 – Экран Settings

Также при выходе из игры текущий прогресс пользователя сохраняется, и у него есть возможность продолжить игру с того момента, где остановился нажатием на кнопку “Load”. Для того чтобы изменить настройки игры необходимо ее перезапустить. При выборе пользователем фигуры для игры (1 – крестики, 2 – нолики) автоматически выбирается и очередь его хода, так как крестики всегда ходят первыми.

При нажатии на “About” открывается справка о создателях игры.

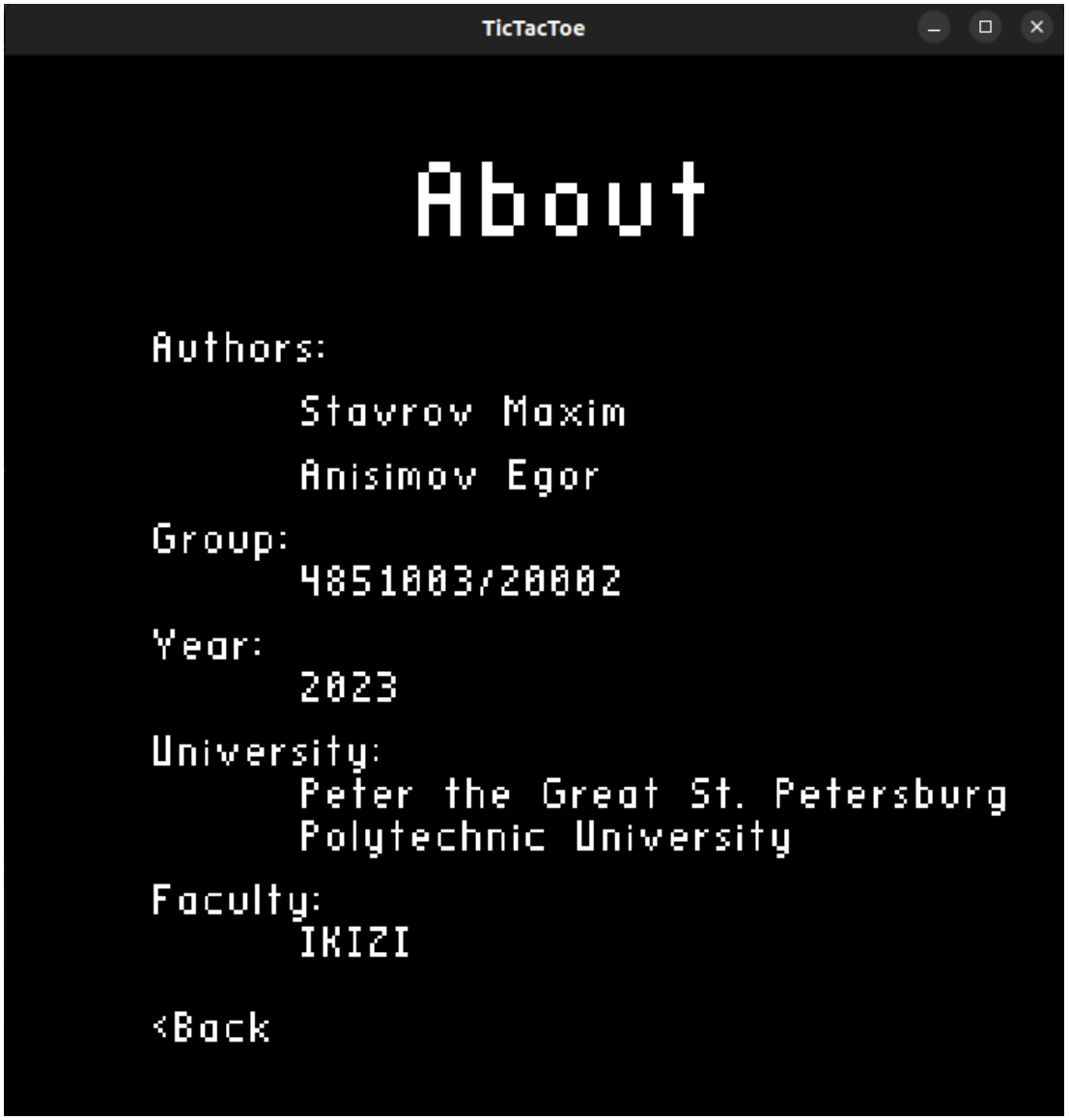


Рисунок 8 – Экран About

3.2 Графика

Для вывода изображения в OpenGL используются графические примитивы, такие как точки, линии и треугольники, которые затем отображаются на экране с помощью растеризации. По умолчанию, графика в OpenGL выстраивается относительно координат, края которых определены размерами окна, и имеют значения в пределах от -1 до 1, а центром координат по умолчанию является середина окна, но его можно сдвинуть.

Функция glTranslatef() – сдвигает объекты вдоль определенного вектора, glSalef() – изменяет размер объектов на определенный множитель и glRotatef() – вращает объекты на определенный угол. Эти функции являются частями матричных преобразований в OpenGL и позволяют изменять положение, размер и ориентацию геометрических примитивов.

3.3 Игровой процесс

Чтобы сделать возможным клик пользователя по свободным клеткам, была реализована функция mouse(int button, int state, int mousex, int mousey). Она принимает координаты курсора мышки и переводит их в экранные координаты. Каждая кнопка имеет занимает определенные координаты, таким образом фиксируется нажатие.

3.4 Сохранение и загрузка

При нажатии на клавишу “Esc” во время игры, пользователь возвращается в главное меню. После этого в специально выделенный файл “data.txt” сохраняются все пользовательские параметры, которые необходимо учитывать при возобновлении игры. Далее при нажатии на кнопку “Load”, данные из файла подгружаются и игра возобновляется.

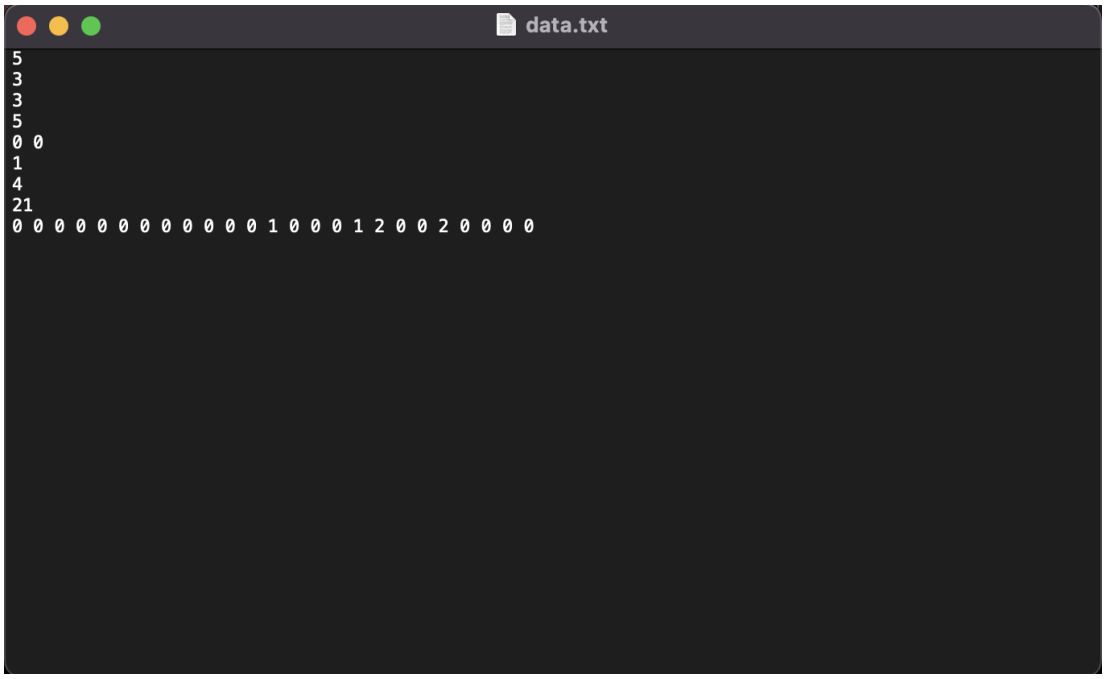


Рисунок 9 – Содержимое файла data.txt

Для упрощения хранения матрицы были введены числовые обозначения состояния каждой клетки поля:

- 0 – Свободное поле;
- 1 – Крестик;
- 2 – Нолик;

3.5 Таблица рекордов

Во время каждой игры происходит подсчет времени. В случае если игрок побеждает, то он может записать свое имя в окне выигрыша. Результат игры записывается в файл records.txt. На экране records отображаются результаты прошлых игр с их нумерацией и именем пользователя, который играл. Если пользователь победил в игре – ему предлагается ввести имя, в противном случае такой возможности у него нет, следовательно его результат не будет попадать в таблицу рекордов.

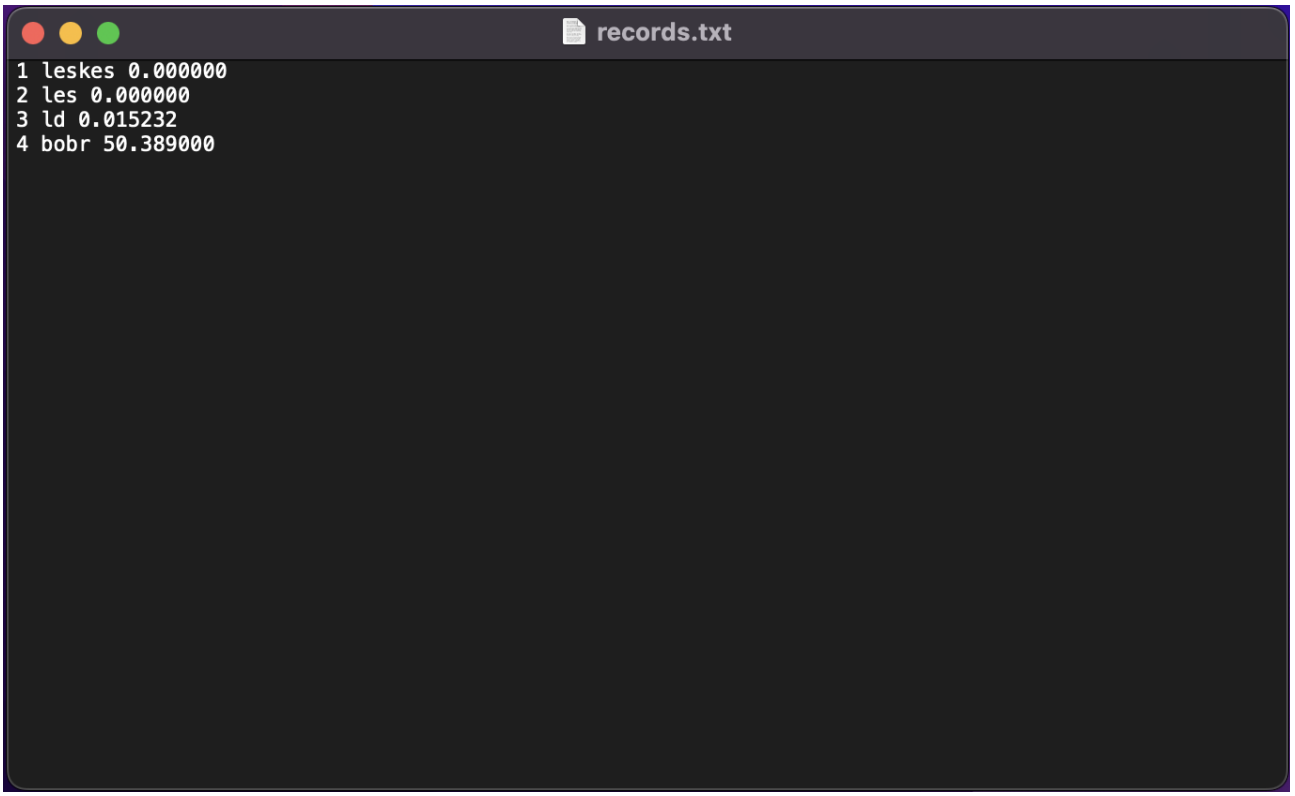


Рисунок 10 – Содержимое файла records.txt

Рисунок 11 – Таблица рекордов

3.6 Алгоритмы

При выборе пользователем фигуры для игры (1 – крестики, 2 – нолики) автоматически выбирается и очередь его хода, т.к. крестики всегда ходят первыми.

Программа передает управление в функцию start_game(). При выборе пользователем крестиков (переменная choice = 1) инициализируются 3 переменные:

- ai_move – фигура для игры компьютером, становится равной CIRCLE;
- player_move – фигура для игры пользователем, становится равной CROSS;
- current_move – текущий ход, становится равным PLAYER;

При выборе ноликов (choice = 2) вышеописанные переменные инициализируются противоположными значениями. Далее весь цикл игры заключается в функции game().

В самом начале возвращается текущее состояние игрового поля:

- Игра закончилась выигрышем крестиков;
- Игра закончилась выигрышем ноликов;
- Ничья;
- Игра продолжается;

В первых трех случаях игра прекращается и выводится соответствующее сообщение, при этом если за выигрышную фигуру игра человек, ему предлагается ввести свое имя для записи в таблицу рекордов.

Если игра продолжается, то программа, в случае значения переменной current_move = PLAYER будет ждать нажатия игроком в свободную клетку для постановки соответствующей ему фигуры. В случае, когда current_move = AI, программа вызывает последовательно функцию ai_move_game(), которая отвечает за генерацию хода компьютера, затем снова функцию game() и функцию отрисовки текущего состояния поля display(). Так как текущий ход может заканчиваться и после ввода фигуры пользователем, то функции game() и

display() вызываются и после регистрации корректной постановки фигуры пользователем в функции mouse().

Функция ai_move_game() отвечает за принятие решения хода компьютером.

В первую очередь создается структура move, содержащая два поля: координаты x и y для хода. Перед тем, как поведение функции будет зависеть от выбранной пользователем сложности, программа проверяет, какой по счету это ход. Если это первый ход компьютера и компьютер ходил первым (amount_moves = 0), то первый ход будет производится в середину видимого игрового поля – это функция get_first_ai_move(). Если это первый ход компьютера и компьютер ходил вторым (amount_moves = 1), то выбор осуществляется в одну из свободных клеток вокруг первого хода пользователя случайным образом – это функция get_random_move(0) с флагом 0.

Далее ход компьютера осуществляется в соответствии со сложностью:

3.6.1 Первый уровень сложности (случайный ход) заключается в следующем: Ход компьютера генерируется случайным образом среди всех свободных клеток (функция get_random_move(1) с флагом 1)**.**

3.6.2 Второй уровень сложности (Блокировка линии игрока) состоит из следующих этапов:

1. Проверить, есть ли у игрока линия длины WIN_LEN-1, которую он может закрыть, чтобы выиграть игру.
2. Если есть, заблокировать линию игрока, сделав ее длину равной WIN_LEN.
3. Иначе, выполнить случайный код.

Проще говоря, компьютер предотвращает немедленный проигрыш, а в остальных случаях ходит случайно.

3.6.3 Третий уровень сложности (поиск выигрышного хода) состоит из следующих этапов:

1. Проверить, есть ли у компьютера линия длины WIN_LEN-1, которую он может закрыть, чтобы выиграть игру.
2. Если есть, сделать ход, замкнув эту линию и выиграв игру.
3. Проверить, есть ли у игрока линия длины WIN_LEN-1, которую он может закрыть, чтобы выиграть игру.
4. Если есть, заблокировать линию игрока 1, сделав ее длину равной WIN_LEN.
5. Иначе, выполнить случайных ход.

Проще говоря, компьютер в первую очередь ищет ход для моментального выигрыша, в противном случае пытается предотвратить немедленный проигрыш и в случае отсутствия вышеперечисленных ситуаций совершает случайный ход.

3.6.4 Четвертый уровень сложности (минимакс) состоит из следующих этапов:

1. Оценить текущее состояние игры и выбрать лучший ход при помощи алгоритма минимакс.
2. Вернуть найденный ход.

Алгоритм минимакс будет просчитывать ходы вперед до тех пор пока не будет достигнута или победа или поражение или ничья. Назовем эти события – терминальным состоянием. Попав в терминальное состояние компьютер начисляет очки: за победу начисляется 10 очков, за поражение -10 и за ничью 0. Вместе с этим, аналогичные расчеты производятся для ходов игрока: компьютер будет выбирать ход с наибольшим счетом, если ходит он сам или ход с наименьшим счетом если ход выполняет человек. Такое правило называется минимаксом.

Кратко алгоритм «минимакса» можно описать как рекурсивную функцию следующего содержания: 1. Функция возвращает счет, если найдено терминальное состояние (победа – 10, проигрыш – 10, ничья – 0 очков). 2. Если

терминального состояния не найдено, функция проходит по всем свободным клеткам поля, делает в них ходы и рекурсивно вызывает функцию «минимакс» от имени оппонента для каждого своего хода. 3. Функция оценивает наилучший счет для текущего игрока на данном этапе и возвращает этот счет. Наилучшим счетом для игрока-человека является счет -10, для игрока компьютера – 10.

В реализованной программе сначала вызывается функция get_move_minimax(player), которая вызывается от имени компьютера. Затем в цикле перебираются все свободные клетки, в которые совершается ход компьютера и вызывается функция minimax(), которая возвращает количество очков для данного хода. Если возвращенное количество очков лучше текущего, то текущая позиция запоминается как наилучшая.

Функция minimax() сначала проверяет, достигнута ли максимальная глубина рекурсии, равная длине выигрышной линии. Если достигнута, то возвращается 0 очков, в противном случае, проверяется достижение терминального состояния. Если такое состояние достигнуто, возвращаем 10 если победил компьютер, -10, если победил человек, 0 – если ничья. Если такого состояния достигнуто не было, то циклом перебираются все свободные клетки поля, в которые совершается ход от имени игрока, переданного в функцию минимакс. Далее вычисляется значение очков рекурсивным вызовом минимакса с противоположным игроком и уменьшением глубины рекурсии. Если текущий игрок - компьютер, то максимизируем текущее значение очков для данной позиции, если текущий игрок - человек, то минимизируем текущее значение очков для данной позиции.

Значение терминального состояние возвращается функцией evaluate(), которая в случае победы компьютера возвращает 10 очков, в случае победы игрока – 10 очков, в случае ничьи 3 очка и в случае, если игра не окончена, 0 очков.

Текущее состояние игрового поля возвращается функцией check_state():

- WIN_CROSS – выигрыш крестиков;
- WIN_CIRCLE – выигрыш ноликов;

- DRAW – ничья;
- GAME_CONT – игра продолжается;

В первую очередь проверяется выигрыш по горизонталям (check_lines()), вертикалям(check_columns()) и диагоналям(check_diag()). Если эти функции не вернули выигрыш ни за одну фигуру, проверяется значение глобальной переменной free_cell, отвечающие за оставшееся количество пустых клеток. Изначально она равна количеству всех клеток на поле и декрементируется каждый раз, когда совершается ход игроком или компьютером. Следовательно, если free_cell = 0, то достигнута ничья, иначе игра продолжается.

Глубина для вышеописанного алгоритма минимакса зависит от нескольких факторов, таких как размер поля, сложность игры и требования к производительности. Увеличение глубины приводит к повышению точности прогнозирования ходов, но за счет более высоких вычислительных затрат и времени выполнения. Учитывая размер поля (FIELD_SIZE) и длину линии для победы (WIN_LEN), есть несколько разумных подходов к выбору глубины:

1. Меньшая глубина(2-4): Это будет делать алгоритм быстрым, но менее точным. Рекомендуется для большего размера поля или когда требуется быстрая отзывчивость, особенно при использовании на слабых устройствах.
2. Средняя глубина(4-6): Предоставляет более точные предсказания и является разумным компромиссом между производительностью и точностью. Рекомендуется для среднего размера поля и варианта выигрышной линии.
3. Большая глубина(6-8 и более): Значительно улучшает точность предсказаний ходов, но может потребовать больше времени и ресурсов для вычислений. Рекомендуется для меньших размеров поля и вариантов выигрышной линии или для сильных вычислительных аппаратных средств.

Однако важно помнить, что эффективность и оптимальная глубина зависят от различных факторов. Единственным способом точно определить оптимальную глубину является экспериментирование и тестирование алгоритма на разных размерах поля и разных размерах выигрышной линии.

4 Статистика алгоритмов

Таблица 1 – Статистика алгоритмов

Размер поля	Алгоритм 1	Алгоритм 2	Алгоритм 3	Алгоритм 4
3	0.000005	0.000004	0.000007	0.000006
5	0.000022	0.000046	0.000036	0.000174
10	0.000025	0.000058	0.000085	0.004074
15	0.000033	0.000103	0.000166	0.033187
25	0.000075	0.000216	0.000353	0.091364

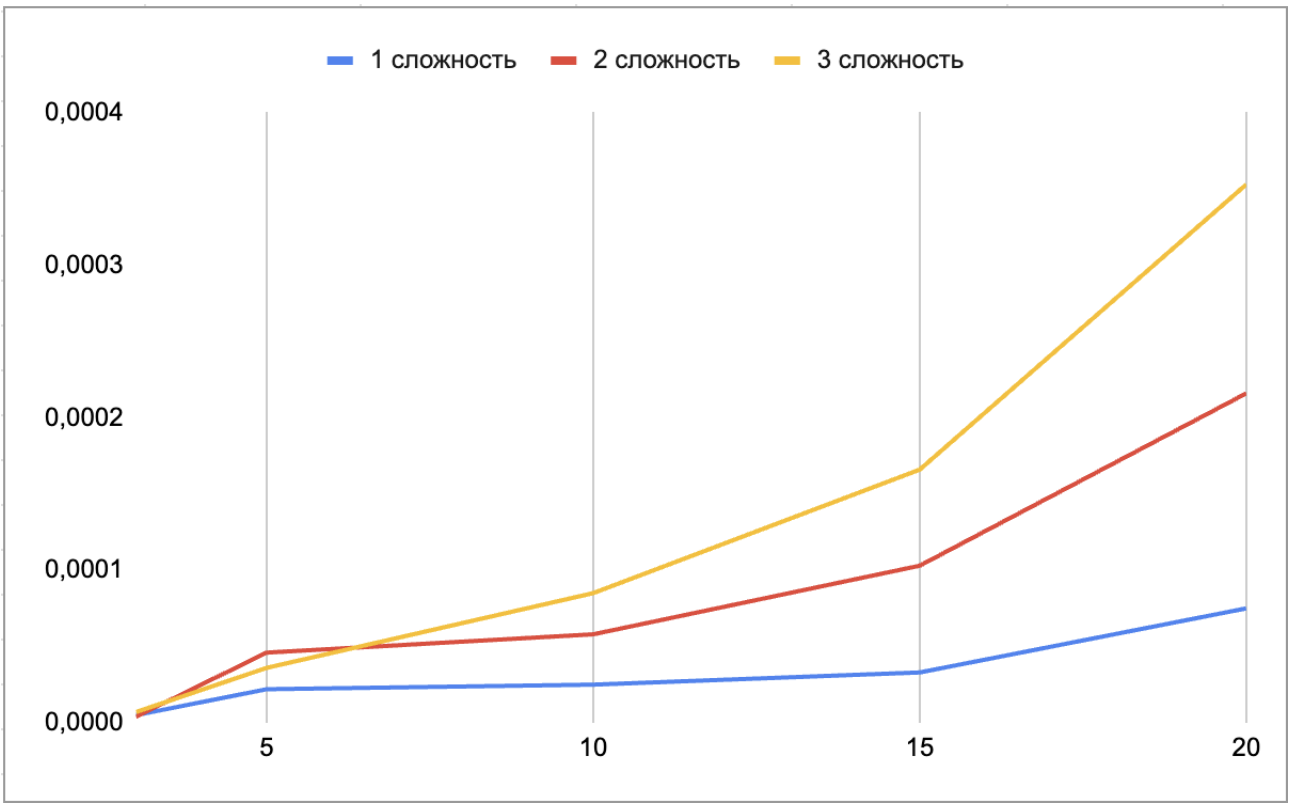


Рисунок 12 – График зависимости времени работы алгоритма от длины выигрышной линии (первые три алгоритма)

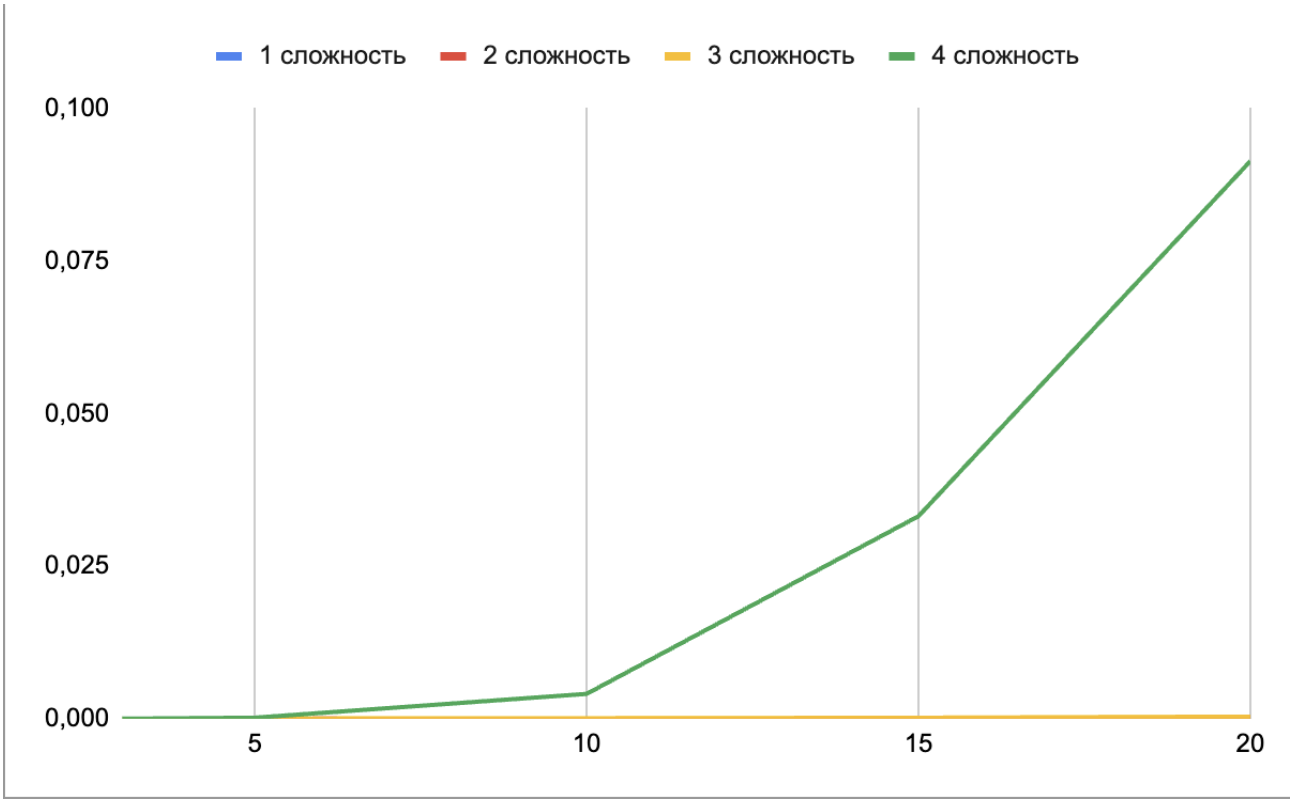


Рисунок 13 – График зависимости времени работы алгоритма от длины выигрышной линии (все алгоритмы)

Заключение

В ходе выполнения данной курсовой работы была реализована игра “Крестики-нолики на бесконечном поле” в соответствии со всеми правилами, описанными в начале работы, а именно – реализовано несколько уровней сложности для игры компьютера. Реализовано “бесконечное поле”, сохранение игры в текущем состоянии с возможностью выхода в главное меню. Реализовано меню, в котором пользователь может указывать размер игрового поля, длину выигрышной линии, уровень сложности, а также знак для игры. В игре реализована таблица рекордов.

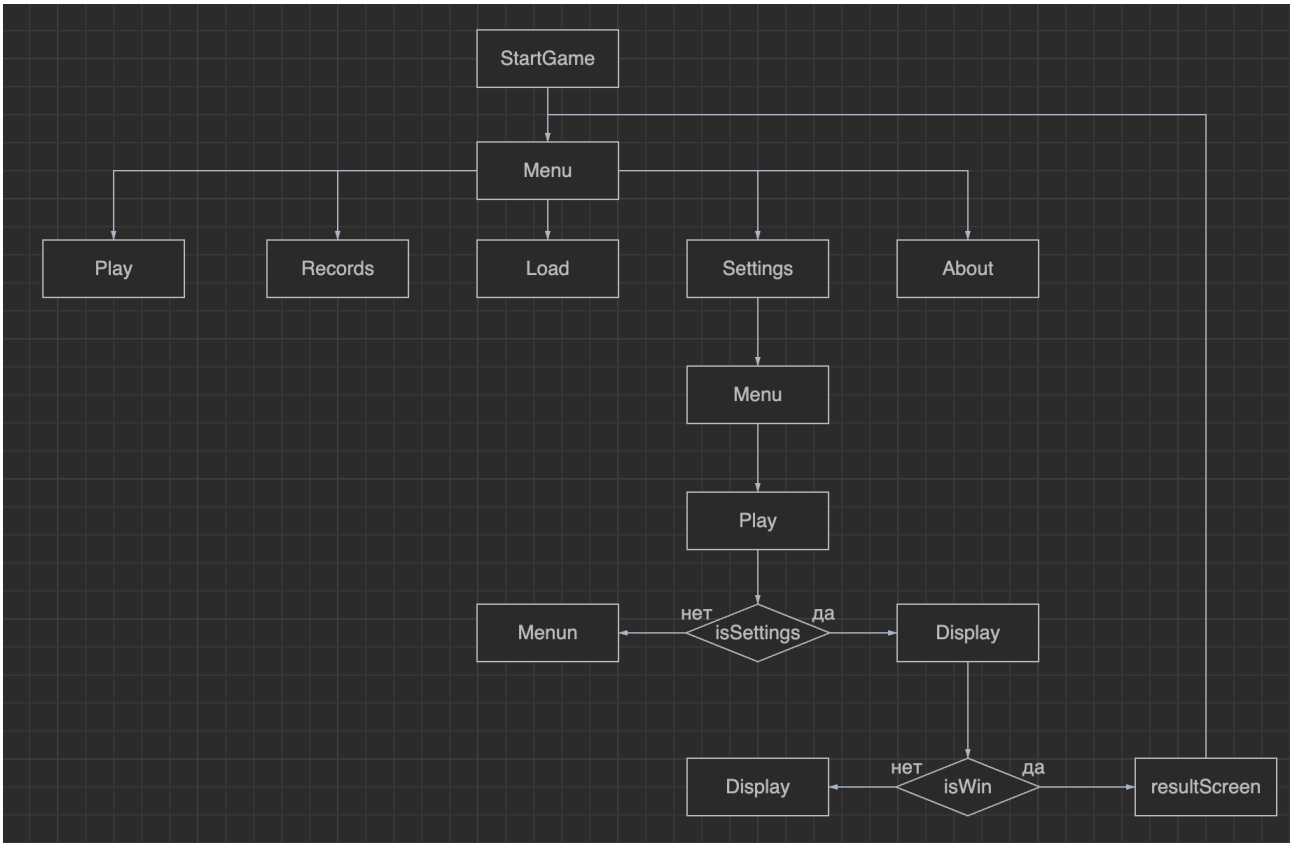
Получены навыки работы с фреймворком OpenGL, а также изучены различные виды алгоритмов для разных уровней сложности компьютера. Например, алгоритм Minimax. Для оптимальной работы программы были использованы были использованы такие структуры данных как структуры, перечисления и динамический массивы.

Список использованной литературы

1. Руководство по OpenGl [opengl-tutorial]. – URL: http://www.opengl-tutorial.org/ru/ (дата обращения 20.03.2023)
2. Алгоритм поиска оптимального решения в игре крестики-нолики – URL: https://dfe.petrsu.ru/koi/teaching/ai/ai_lab1.pdf (дата обращения 12.04.2023)
3. Реализация алгоритма минимакс на примере игры крестики-нолики – URL: https://habr.com/ru/articles/329058/ (дата обращения 03.05.2023)
4. Руководство по работе с графикой в OpenGL – URL: http://grafika.me/node/130 (дата обращения 06.03.2023)

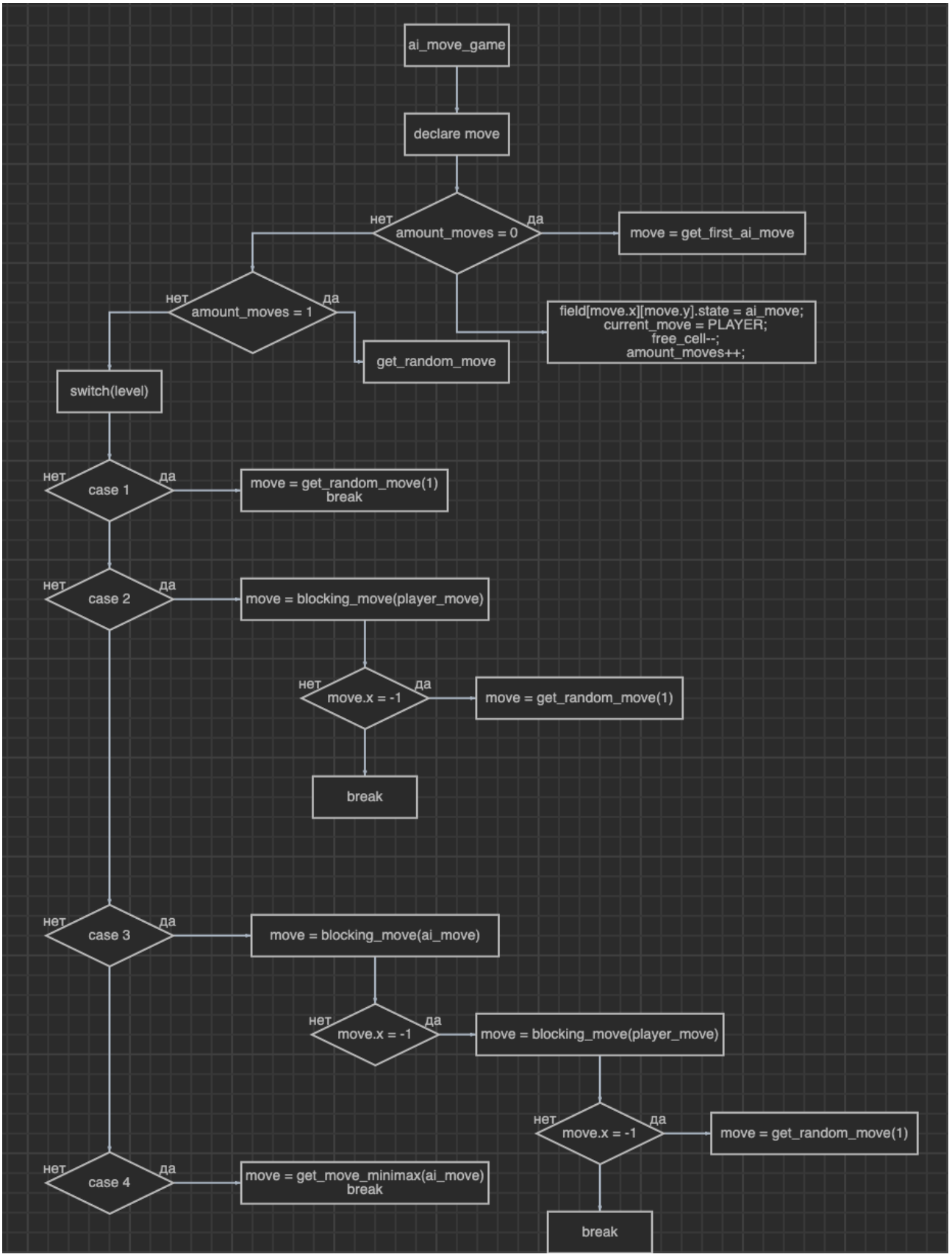
ПРИЛОЖЕНИЕ А

Блок-схема общей работы приложения



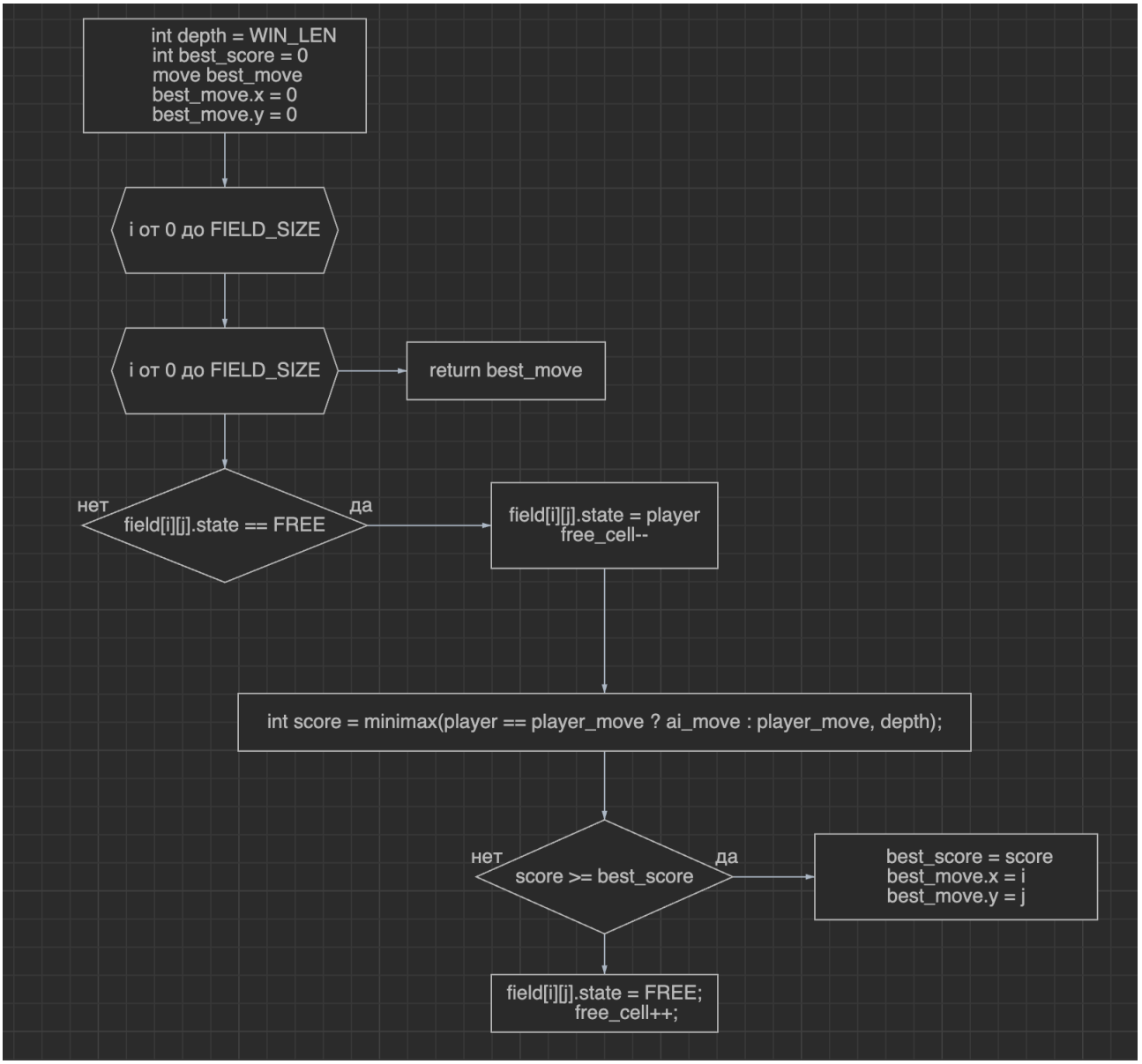
ПРИЛОЖЕНИЕ Б

Блок-схема процедуры ai_move_game



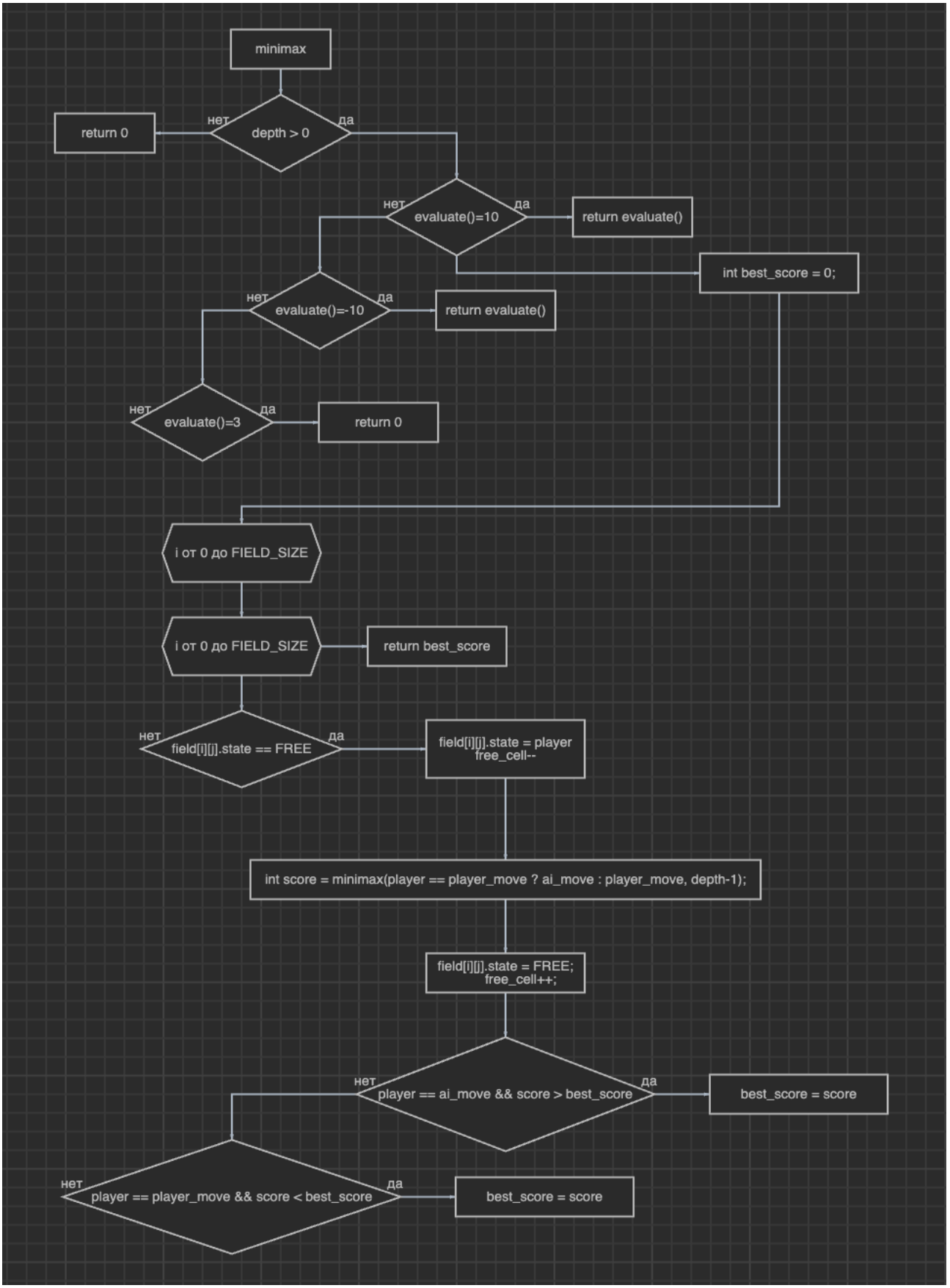
ПРИЛОЖЕНИЕ В

Блок-схема функции get_move_minimax



ПРИЛОЖЕНИЕ Г

Блок-схема функции minimax



ЛИСТИНГ ПРОГРАММЫ

```
#include <GL/glut.h>
#include <GL/gl.h>

#include "stb_easy_font.h"

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>
#include <time.h>

#define MIN_SIZE_FIELD 3
#define MAX_SIZE_FIELD 10000
#define MAX_VIEW_SIZE 10
#define WINDOW_SIZE 700

typedef struct{
    int x;
    int y;
}move;
enum move{
    PLAYER,
    AI
};
enum state_ceil{
    FREE = 0,
    CROSS = 1,
    CIRCLE = 2
};
enum state_game{
    GAME_CONT = 0,
```

```
    WIN_CROSS = 1,
    WIN_CIRCLE = 2,
    DRAW = 3
};
typedef struct{
    float c_x;
    float c_y;
    enum state_ceil state;
}Cell;

clock_t t1, t2 = 0;

int WIN_LEN;
int FIELD_SIZE;
int VIEW_SIZE;

char fieldSize[10000];
char winLen[10000];
char diffLevel[10];
char whoStart[10];
char userName[100];

int flagStart = 0;
int flagSettings = 0;
int flagAbout = 0;
int flagWin = 0;
int flagPause = 1;
int flagRecords = 0;
int playerWin = 0;
int settingsChoise = 0;
int current_x = 0;
int current_y = 0;
int LVL;
int choice;

enum state_game state_game = GAME_CONT;
```

```
enum state_ceil current_move;
enum state_ceil ai_move;
enum state_ceil player_move;
unsigned int amount_moves;
unsigned int free_cell;

Cell **field;
Cell **view_field;

void game(void);
enum state_game check_state(void);
void display(void);


Cell ** get_dynamic_array(void){
    int i;
    Cell **array = (Cell**)malloc(sizeof(Cell*)*FIELD_SIZE);
    assert(array);
    for (i = 0; i < FIELD_SIZE; ++i){
     array[i] = (Cell*)malloc(sizeof(Cell)*FIELD_SIZE);
     assert(array[i]);
    }
    return array;
}

void init(void){

    int i, j;
    if(FIELD_SIZE <= MAX_VIEW_SIZE)
     VIEW_SIZE = FIELD_SIZE;
    else{
     VIEW_SIZE = MAX_VIEW_SIZE;
     current_x = (FIELD_SIZE - VIEW_SIZE)/2;
     current_y = (FIELD_SIZE - VIEW_SIZE)/2;
    }
```

```
    amount_moves = 0;
    free_cell = FIELD_SIZE*FIELD_SIZE;

    field = get_dynamic_array();
    view_field = get_dynamic_array();

    for (i = 0; i < FIELD_SIZE; ++i){
     for (j = 0; j < FIELD_SIZE ; ++j){
          field[i][j].c_x = i+0.5;
          field[i][j].c_y = j+0.5;
          field[i][j].state = FREE;
     }
    }
    for (i = 0; i < VIEW_SIZE; ++i){
     for (j = 0; j < VIEW_SIZE; ++j){
          view_field[i][j].c_x = i+0.5;
          view_field[i][j].c_y = j+0.5;
          view_field[i][j].state = FREE;
     }
    }
}

void start_game(void){
    if(choice == 1){
     ai_move = CIRCLE;
     player_move = CROSS;
     current_move = PLAYER;
    }
    else if(choice == 2){
     ai_move = CROSS;
     player_move = CIRCLE;
     current move = AI;
     game();
    }
}
```

```
      enum state_game check_lines(void){
          int i, j, k;
          enum state_game player;
          for (i = 0; i < FIELD_SIZE; i++) {
           for (j = 0; j <= FIELD_SIZE - WIN_LEN; j++) {
                if(field[i][j].state == CROSS || field[i][j].state
== CIRCLE){
                     player = field[i][j].state;
                     for (k = 1; k < SIN_LEN; k++) {
                          if (field[i][j + k].state != player) {
                                   break;
                          }
                     }
                     if (k == WIN_LEN) {
                          return player;
                     }
                }
           }
          }
          return 0;
      }

      enum state_game check_columns(void){
          int i, j, k;
          enum state_game player;
          for (j = 0; j < FIELD_SIZE; j++) {
           for (i = 0; i <= FIELD_SIZE - WIN_LEN; i++) {
                if(field[i][j].state == CROSS || field[i][j].state
== CIRCLE){
                     player = field[i][j].state;
                     for (k = 1; k < SIN_LEN; k++) {
                          if (field[i + k][j].state != player) {
                               break;
                          }
                     }
                     if (k == WIN_LEN) {
```

```
                    return player;
                }
            }
        }
        return 0;
    }

    enum state_game check_diag(void){
        int i, j, k;
        enum state_game player;
        for (i = 0; i <= FIELD_SIZE - WIN_LEN; i++) {
         for (j = 0; j <= FIELD_SIZE - WIN_LEN; j++) {
             if(field[i][j].state == CROSS || field[i][j].state
== CIRCLE){
                 player = field[i][j].state;
                 for (k = 1; k < SIN_LEN; k++) {
                     if (field[i + k][j + k].state != player)
{
                         break;
                     }
                 }
                 if (k == WIN_LEN) {
                     return player;
                 }
             }
              if(field[i][j + WIN_LEN - 1].state == CROSS ||
field[i][j + WIN_LEN - 1].state == CIRCLE){
                 player = field[i][j + WIN_LEN - 1].state;
                 for (k = 1; k < SIN_LEN; k++) {
                         if (field[i + k][j + WIN_LEN - 1 -
k].state != player) {
                             break;
                         }
                 }
                 if (k == WIN_LEN) {
```

```
                    return player;
                }
            }
        }
    }
    return 0;
}

enum state_game check_state(void){
    enum state_game state;
    state = check_lines();
    if(state)
        return state;
    state = check_columns();
    if(state)
        return state;
    state = check_diag();
    if(state)
        return state;
    int c = 0;
    for (int i = 0; i < FIELD_SIZE; ++i){
        for (int j = 0; j < FIELD_SIZE; ++j){
            if(field[i][j].state != FREE)
                c++;
        }
    }
    if(free_cell == 0)
    return DRAW;
    else
    return GAME CONT;
}

void drawLine(float x1, float y1, float x2, float y2){
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
}
```

```
void drawField(void){
    glBegin(GL_QUAD_STRIP);
    glColor3f(0.9, 0.9, 0.9);
    glVertex2f(0, VIEW_SIZE);
    glVertex2f(VIEW_SIZE, VIEW_SIZE);
    glVertex2f(0, 0);
    glVertex2f(VIEW_SIZE, 0);
    glEnd();

    glLineWidth(2);
    glBegin(GL_LINES);
    glColor3f(0,0,0);
    for(int i = 1; i < VIEW_SIZE; i++){
     drawLine(0,i,VIEW_SIZE,i);
     drawLine(i,0,i,VIEW_SIZE);
    }
    glEnd();
}

void drawCross(float c_x, float c_y){
    glLineWidth(8);
    glBegin(GL_LINES);
    glColor3f(0,0,1);
    drawLine(c_x-0.3, c_y-0.3,c_x+0.3,c_y+0.3);
    drawLine(c_x+0.3,c_y-0.3,c_x-0.3,c_y+0.3);
    glEnd();
}

void drawCircle(float c_x, float c_y){
    glLineWidth(10);
    glBegin(GL_TRIANGLE_STRIP);
    glColor3f(1,0,0);

    glVertex2f( c_x-0.3, c_y );
     for(int i = 0; i <= 360; i++ ) {
```

```
                float a = (float)i / 360 * 3.1415f * 65;
                glVertex2f( c_x-cos( a ) * 0.3, c_y-sin( a ) * 0.3
);
            }

        glEnd();
    }

    void print_string(float x, float y, char *text, float r,
float g, float b) {
        static char buffer[99999]; // ~500 chars
        int num_quads;

            num_quads = stb_easy_font_print(x, y, text, NULL,
buffer, sizeof(buffer));

        glColor3f(r,g,b);
        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(2, GL_FLOAT, 16, buffer);
        glDrawArrays(GL_QUADS, 0, num_quads*4);
        glDisableClientState(GL_VERTEX_ARRAY);
    }

    void settingsScreen(void) {
        glutReshapeWindow(WINDOW_SIZE, WINDOW_SIZE);
        glClear(GL_COLOR_BUFFER_BIT);
        glPushMatrix();
        glScalef(0.02, -0.02, 1);
        print_string(-20, -40,"Settings", 1, 1, 1);
        glScalef(50, -50, 1);

        glScalef(0.01, -0.01, 1);
            print_string(-80, -40, "Enter field size: ", 1,
settingsChoise == 0 ? 0 : 1, 1);
        glScalef(100, -100, 1);
```

```
        glScalef(0.01, -0.01, 1);
        print_string(12, -40, fieldSize, 1, 1, 1);
        glScalef(100, -100, 1);

        glScalef(0.01, -0.01, 1);
             print_string(-80,  -20,  "Enter  win  len:  ",  1,
settingsChoise == 1 ? 0 : 1, 1);
        glScalef(100, -100, 1);

        glScalef(0.01, -0.01, 1);
        print_string(0, -20, winLen, 1, 1, 1);
        glScalef(100, -100, 1);

        glScalef(0.01, -0.01, 1);
         print_string(-80, 0, "Enter difficulty level(1-4): ", 1,
settingsChoise == 2 ? 0 : 1, 1);
        glScalef(100, -100, 1);

        glScalef(0.01, -0.01, 1);
        print_string(70, 0, diffLevel, 1, 1, 1);
        glScalef(100, -100, 1);

        glScalef(0.01, -0.01, 1);
          print_string(-80, 20, "Enter who will start(1-x, 2-o):
", 1, settingsChoise == 3 ? 0 : 1, 1);
        glScalef(100, -100, 1);

        glScalef(0.01, -0.01, 1);
        print_string(90, 20, whoStart, 1, 1, 1);
        glScalef(100, -100, 1);

        glScalef(0.01, -0.01, 1);
        print_string(-80, 50, "<Back", 1, 1, 1);
        glScalef(100, -100, 1);

        glPopMatrix();
```

```
        glutSwapBuffers();
    }

    void aboutScreen(void) {
        glutReshapeWindow(WINDOW_SIZE, WINDOW_SIZE);
        glClear(GL_COLOR_BUFFER_BIT);
        glPushMatrix();

        glScalef(0.02, -0.02, 1);
        print_string(-11, -40,"About", 1, 1, 1);
        glScalef(0.4, 0.4, 1);
        print_string(-90, -60,"Authors: ", 1, 1, 1);
        print_string(-55, -30,"Anisimov Egor", 1, 1, 1);
        print_string(-55, -45,"Stavrov Maxim", 1, 1, 1);

        print_string(-90, -15,"Group: ", 1, 1, 1);
        print_string(-55, -5,"4851003/20002", 1, 1, 1);

        print_string(-90, 10,"Year: ", 1, 1, 1);
        print_string(-55, 20,"2023", 1, 1, 1);

        print_string(-90, 35,"University: ", 1, 1, 1);
         print_string(-55, 45,"Peter the Great St. Petersburg",
1, 1, 1);
        print_string(-55, 55,"Polytechnic University", 1, 1, 1);

        print_string(-90, 70,"Faculty: ", 1, 1, 1);
        print_string(-55, 80,"IKIZI", 1, 1, 1);

        print_string(-90, 100,"<Back", 1, 1, 1);

        glScalef(125, -125, 1);

        glPopMatrix();
        glutSwapBuffers();
    }
```

```
void recordsScreen(void) {
    glutReshapeWindow(WINDOW_SIZE, WINDOW_SIZE);
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();

    glScalef(0.02, -0.02, 1);
    print_string(-18, -40,"Records", 1, 1, 1);
    glScalef(0.4, 0.4, 1);
    FILE* r;
    r = fopen("records.txt", "r");
    int s = -60;
    char string[31] = { '\0' };
    int d = 0;
    while(fgets(string, 30, r)) {
     print_string(-60, s, string, 1, 1, 1);
     s += 20;
     memset(string, '\0', 30);
    }

    print_string(-90, 100,"<Back", 1, 1, 1);

    glScalef(125, -125, 1);
    glPopMatrix();
    glutSwapBuffers();
}

void menuScreen(void) {
    glutReshapeWindow(WINDOW_SIZE, WINDOW_SIZE);
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();

    glScalef(0.02, -0.02, 1);
    print_string(-11, -40,"Menu", 1, 1, 1);
    glScalef(0.5, 0.5, 1);
    print_string(-12, 20, "Load", 1, 1, 1);
```

```
        print_string(-20, 40, "Records", 1, 1, 1);
        print_string(-20, 60, "Settings", 1, 1, 1);
        print_string(-15, 80, "About", 1, 1, 1);
        glScalef(200, -200, 1);

        glScalef(0.1, 0.1, 1);
        glBegin(GL_LINE_STRIP);
        glVertex2f(-0.5, 1.0);
        glVertex2f(-0.5, -0.5);
        glVertex2f(0.5, 0.2);
        glVertex2f(-0.5, 1);
        glEnd();
        glScalef(5, 5, 1);

        glPopMatrix();
        glutSwapBuffers();
    }

    void alertScreen(void){
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        glPushMatrix();

        glScalef(0.5, 0.5, 1);
           print_string(-8,  -60,state_game  ==  DRAW  ?  "Draw"  :
"Win", 1, 1, 1);
        glScalef(0.8, 0.8, 1);
        if (check_state()==player_move){
        print_string(-80, -55, "Enter your name: ", 1, 1, 1);
        print_string(15, -55, userName, 1, 1, 1);
    }
        print_string(-60, 60,"<Back", 1, 1, 1);
        glScalef(2.5, 2.5, 1);

        glScalef(50, 50, 1);
        if(state_game == WIN_CIRCLE) {
```

```
            drawCircle(0, 0);
        } else if(state_game == WIN_CROSS){
            drawCross(0, 0);
        }

        glScalef(0.02, 0.02, 1);
        glPopMatrix();
        glutSwapBuffers();
    }

    void fill_view_field(void){
        for (int i = 0; i < VIEW_SIZE; ++i){
            for (int j = 0; j < VIEW_SIZE; ++j){
                int x = current_x + i;
                int y = current_y + j;
                view_field[i][j].state = field[x][y].state;
                view_field[i][j].state = field[x][y].state;
            }
        }
    }

    move get_first_ai_move(void){
        move move;
        for (int i = 0; i < FIELD_SIZE; ++i){
            for (int j = 0; j < FIELD_SIZE; ++j){
                if(i == current_x + VIEW_SIZE/2 && j == current_y
+ VIEW_SIZE/2){
                    move.x = i;
                    move.y = j;
                }
            }
        }
        return move;
    }

    int check_coordinate_in_field(int x, int y){
```

```
            return (x >= 0 && x < FIELD_SIZE && y >= 0 && y <
FIELD_SIZE);
      }

      move get_random_move(int flag){
          int count_move = 0;
          if(flag){
           for (int i = 0; i < FIELD_SIZE; ++i){
                for (int j = 0; j < FIELD_SIZE; ++j){
                     if(field[i][j].state == FREE){
                          count_move++;
                     }
                }
           }
          }
          else{
           for (int i = 0; i < FIELD_SIZE; ++i){
                for (int j = 0; j < FIELD_SIZE; ++j){
                     if(field[i][j].state == player_move){
                           if((i  ==  0  &&  j  ==  0)  ||  (i  ==
FIELD_SIZE-1  &&  j  ==  FIELD_SIZE-1)  ||  (i  ==  0  &&  j  ==
FIELD_SIZE-1) || (i == FIELD_SIZE-1 && j == 0))
                                count_move = 3;
                           else if(i == 0 || i == FIELD_SIZE-1 || j
== 0 || j == FIELD_SIZE-1)
                                count_move = 5;
                           else
                                count_move = 8;
                     }
                }
           }
          }
                                    int       **choice_coord       =
(int**)malloc(sizeof(int*)*count_move);
          assert(choice_coord);
          for (int i = 0; i < count_move; ++i){
```

```
 choice_coord[i] = (int*)malloc(sizeof(int)*2);
 assert(choice_coord[i]);
}
int k = 0;
for (int i = 0; i < FIELD_SIZE; ++i){
 for (int j = 0; j < FIELD_SIZE; ++j){
      if(flag){
          if(field[i][j].state == FREE){
                choice_coord[k][0] = i;
                choice_coord[k][1] = j;
                k++;
          }
      }

      else{
           if(field[i][j].state == player_move){
                if(check_coordinate_in_field(i,j+1)){
                     choice_coord[k][0] = i;
                     choice_coord[k][1] = j+1;
                     k++;
                }
                if(check_coordinate_in_field(i+1,j+1)){
                     choice_coord[k][0] = i+1;
                     choice_coord[k][1] = j+1;
                     k++;
                }
                if(check_coordinate_in_field(i+1,j)){
                     choice_coord[k][0] = i+1;
                     choice_coord[k][1] = j;
                     k++;
                }
                if(check_coordinate_in_field(i+1,j-1)){
                     choice_coord[k][0] = i+1;
                     choice_coord[k][1] = j-1;
                     k++;
                }
```

```
                    if(check_coordinate_in_field(i,j-1)){
                        choice_coord[k][0] = i;
                        choice_coord[k][1] = j-1;
                        k++;
                    }
                    if(check_coordinate_in_field(i-1,j-1)){
                        choice_coord[k][0] = i-1;
                        choice_coord[k][1] = j-1;
                        k++;
                    }
                    if(check_coordinate_in_field(i-1,j)){
                        choice_coord[k][0] = i-1;
                        choice_coord[k][1] = j;
                        k++;
                    }
                    if(check_coordinate_in_field(i-1,j+1)){
                        choice_coord[k][0] = i-1;
                        choice_coord[k][1] = j+1;
                        k++;
                    }
                }
            }

        }
    }

    srand(time(NULL));
    int c = rand() % count_move;
    move move;
    move.x = choice_coord[c][0];
    move.y = choice_coord[c][1];
    return move;
}

move blocking_move(enum state_ceil player){
    move move;
```

```
          move.x = -1;
          move.y = -1;
          for (int i = 0; i < FIELD_SIZE; ++i){
           for (int j = 0; j < FIELD_SIZE; ++j){
                     if(field[i][j].state      ==      FREE      &&
check_coordinate_in_field(i,j)){
                     field[i][j].state = player;
                     if(check_state() == player){
                          move.x = i;
                          move.y = j;
                     }
                     field[i][j].state = FREE;
                }
           }
          }
          return move;
      }

      int evaluate(void){
          if(check_state() == ai_move){
           return 10;
          }
          else if(check_state() == player_move){
           return -10;
          }
          else if(check_state() == GAME_CONT){
           return 0;
          }
          else{
           return 3;
          }
      }

      int  minimax(enum  state_ceil  player,  int  depth,  int  alpha,
int beta){
          if (depth>0){
```

```
            if (evaluate()==10){
                 return evaluate();
            }
            else if (evaluate()==-10){
                 return evaluate();
            }
            else if (evaluate()==3){
                 return 0;
            }
            int best_score = 0;
            for (int i = 0; i < FIELD_SIZE; ++i){
                 for (int j = 0; j < FIELD_SIZE; ++j){
                      if(field[i][j].state == FREE){
                           field[i][j].state = player;
                           free_cell--;
                             int   score   =   minimax(player   ==
player_move ? ai_move : player_move, depth-1, alpha, beta);
                           field[i][j].state = FREE;
                           free_cell++;
                            if(player   ==   ai_move   &&   score   >
best_score){
                                best_score = score;
                                alpha = fmax(alpha, best_score);
                                if(beta <= alpha)
                                     break;
                           }
                           else if(player == player_move && score <
best_score){
                                best_score = score;
                                beta = fmin(beta, best_score);
                                if(beta <= alpha)
                                     break;
                           }
                      }
                 }
            }
```

```
            return best_score;
        }
        else return 0;
    }

    move get_move_minimax(enum state_ceil player){
        int depth = WIN_LEN;
        int best_score = 0;
        move best_move;
        best_move.x = 0;
        best_move.y = 0;
        for (int i = 0; i < FIELD_SIZE; ++i){
            for (int j = 0; j < FIELD_SIZE; ++j){
                if(field[i][j].state == FREE){
                    field[i][j].state = player;
                    free_cell--;
                    int alpha = -10;
                    int beta = 10;
                    int score = minimax(player == player_move ?
ai_move : player_move, depth, alpha, beta);
                    if(score >= best_score){
                        best_score = score;
                        best_move.x = i;
                        best_move.y = j;
                    }
                    field[i][j].state = FREE;
                    free_cell++;
                }
            }
        }
        return best_move;
    }

    void ai_move_game(void){
        move move;
        if(amount_moves == 0){
```

```
     move = get_first_ai_move();
    }
    else if(amount_moves == 1){
     move = get_random_move(0);
    }
    else{
     switch(LVL){
          case 1:
               move = get_random_move(1);
               break;
          case 2:
               move = blocking_move(player_move);
               if(move.x == -1)
                    move = get_random_move(1);
               break;
          case 3:
               move = blocking_move(ai_move);
               if(move.x == -1){
                    move = blocking_move(player_move);
                    if(move.x == -1)
                         move = get_random_move(1);
               }
               break;
          case 4:
               move = get_move_minimax(ai_move);
               break;
     }
    }
    field[move.x][move.y].state = ai_move;
    current_move = PLAYER;
    free_cell--;
    amount_moves++;
}

void game(void){
    switch(check_state()){
```

```
                case WIN_CROSS:
                     state_game = WIN_CROSS;
                     glPushMatrix();
                       glTranslatef(VIEW_SIZE*0.5,   VIEW_SIZE*0.5,
0);
                      glScalef(VIEW_SIZE*0.014,   -VIEW_SIZE*0.014,
1);
                     flagWin = 1;
                     flagStart = 0;
                     t2 = clock();
                     glutDisplayFunc(alertScreen);
                     alertScreen();
                     break;
                case WIN_CIRCLE:
                     state_game = WIN_CIRCLE;
                     glPushMatrix();
                       glTranslatef(VIEW_SIZE*0.5,   VIEW_SIZE*0.5,
0);
                      glScalef(VIEW_SIZE*0.014,   -VIEW_SIZE*0.014,
1);
                     flagWin = 1;
                     flagStart = 0;
                     t2 = clock();
                     glutDisplayFunc(alertScreen);
                     alertScreen();
                     break;
                case DRAW:
                     state_game = DRAW;
                     glPushMatrix();
                       glTranslatef(VIEW_SIZE*0.5,   VIEW_SIZE*0.5,
0);
                      glScalef(VIEW_SIZE*0.015,   -VIEW_SIZE*0.015,
1);
                     flagWin = 1;
                     flagStart = 0;
                     glutDisplayFunc(alertScreen);
```

```
                    alertScreen();
                    break;
               case GAME_CONT:
                    if(current_move == AI){
                         ai_move_game();
                         display();
                         game();
                    }
                    break;
          }
          display();
     }

     void display(void){
          glutReshapeWindow(WINDOW_SIZE, WINDOW_SIZE);

          if(flagWin != 1) {
           drawField();
           fill_view_field();
          }

          for (int i = 0; i < VIEW_SIZE; ++i){
           for (int j = 0; j < VIEW_SIZE; ++j){
                glPushMatrix();
                switch(view_field[i][j].state){
                    case CROSS:
                         if(flagWin != 1)

drawCross(view_field[i][j].c_x,field[i][j].c_y);
                         break;
                    case CIRCLE:
                         if(flagWin != 1)

drawCircle(view_field[i][j].c_x,field[i][j].c_y);
                         break;
               }
```

```
            glPopMatrix();
        }
    }

    glFlush();
    glutSwapBuffers();
}

void keyboard1(int button, int x, int y){
    if(flagStart == 1 && flagSettings == 0) {
     switch(button){
          case GLUT_KEY_LEFT:
               if(current_x > 0)
                    current_x--;
               break;
          case GLUT_KEY_RIGHT:
               if(current_x < FIELD_SIZE - VIEW_SIZE)
                    current_x++;
               break;
          case GLUT_KEY_UP:
               if(current_y < FIELD_SIZE - VIEW_SIZE)
                    current_y++;
               break;
          case GLUT_KEY_DOWN:
               if(current_y > 0)
                    current_y--;
               break;
     }
     display();
    }
    else if(flagStart == 0 && flagSettings == 1){
     if(button == GLUT_KEY_DOWN) {
          if(settingsChoise <= 3) {
               settingsChoise += 1;
          } else {
               settingsChoise = 0;
```

```
                }
            }
            if(button == GLUT_KEY_UP) {
                if(settingsChoise >= 0) {
                    settingsChoise -= 1;
                } else {
                    settingsChoise = 3;
                }
            }
            settingsScreen();
        }
    }
    void load() {
        init();
        FILE *fp = fopen("data.txt", "r");
        char o = '\0';
            fscanf(fp, "%d\n%d\n%d\n%d\n%d %d\n%u\n%d\n%d\n",
&FIELD_SIZE, &WIN_LEN, &LVL, &VIEW_SIZE, &current_x, &current_y,
&player_move, &amount_moves, &free_cell);
        for (int k = 0; k < FIELD_SIZE; k++) {
        for (int j = 0; j < FIELD_SIZE; j++) {
            char o;
                fscanf(fp, "%c ", &o);
                if (o == '0') {
                field[k][j].state = FREE;
                }
                else if (o == '1') {
                field[k][j].state = CROSS;
            }
            else if (o == '2'){
                field[k][j].state = CIRCLE;
            }
        }

        }
        fclose(fp);
```

```
}
void save_record() {
     FILE *f = fopen("records.txt", "r+");
     float ftime = ((float)(t2-t1)/CLOCKS_PER_SEC)*1000;
     char names[10][20] = { '\0' };
     float times[10] = { 0.0 };
     char copy[20];
     short int v = 0;
     char c;
     int d=0;
     c = fgetc(f);
     while (c != EOF) {
          fseek(f, -1, SEEK_CUR);
          fscanf(f, "%d %s %s\n", &d, names[v], copy);
          times[v] = atof(copy);
          v++;
          c = fgetc(f);
     }
     int i;
     for (i = v; i > 0 && times[i - 1] > ftime; i--) {
          strcpy(names[i], names[i - 1]);
          times[i] = times[i - 1];
     }
     strcpy(names[i], userName);
     times[i] = ftime;
     fclose(f);
     f = fopen("records.txt", "w");
     for (int n = 0; names[n][0] != '\0'&& n < 5; n++) {
          fprintf(f, "%d %s %f\n", n+1, names[n], times[n]);
     }
     fclose(f);
}
void keyboard2(unsigned char button, int x, int y){
    if(flagStart == 0 && flagSettings == 1) {
     if(button != 13) {
          int userButton = 0;
```

```
                if(button >= 48 && button <= 57 && strlen(winLen)
< 10000 && strlen(fieldSize) < 10000) {

                    if(settingsChoise == 0) {
                              fieldSize[strlen(fieldSize)]      =
(char)button;
                    }
                    if(settingsChoise == 1) {
                         winLen[strlen(winLen)] = (char)button;
                    }
                    if(button == 49 || button == 50 || button ==
51 || button == 52) {
                              if(settingsChoise     ==     2     &&
strlen(diffLevel) < 1) {
                                   diffLevel[strlen(diffLevel)]    =
(char)button;
                         }
                    }
                    if(button == 49 || button == 50) {
                              if(settingsChoise     ==     3     &&
strlen(whoStart) < 1) {
                                   whoStart[strlen(whoStart)]     =
(char)button;
                         }
                    }
               }
               if(settingsChoise == 0 && button == 8) {
                    fieldSize[strlen(fieldSize)-1] = '\0';
               }
               if(settingsChoise == 1 && button == 8) {
                    winLen[strlen(winLen)-1] = '\0';
               }
               if(settingsChoise == 2 && button == 8) {
                    diffLevel[strlen(diffLevel)-1] = '\0';
               }
```

```
                if(settingsChoise == 3 && button == 8) {
                     whoStart[strlen(whoStart)-1] = '\0';
                }
                settingsScreen();
           }
          }
                  if(flagWin   ==   1   &&   flagStart   ==   0   &&
player_move==check_state()) {
           if(button == 8) {
                userName[strlen(userName)-1] = '\0';
                alertScreen();
           }
           else if (button==13){
           save_record();
           flagWin = 0;
                flagPause = 1;
                glClear(GL_COLOR_BUFFER_BIT);
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                glScalef(VIEW_SIZE/2.0, VIEW_SIZE/2.0, 1);
                                glScalef(1/(VIEW_SIZE*0.014),
1/(-VIEW_SIZE*0.014), 1);

                glutDisplayFunc(menuScreen);
                menuScreen();
          }
           else {
                userName[strlen(userName)] = (char)button;
                alertScreen();
           }

          }
          // pause
          if(flagWin == 0 && flagPause == 0 && button == 27) {
           flagPause = 1;

           FILE *file = fopen("data.txt", "w");
```

```
            fprintf(file,  "%d\n%d\n%d\n%d\n%d  %d\n%d\n%d\n%d\n",
FIELD_SIZE,  WIN_LEN,  LVL,  VIEW_SIZE,  current_x,  current_y,
player_move, amount_moves, free_cell);
            for (int i = 0; i < FIELD_SIZE; i++) {
                    for (int j = 0; j < FIELD_SIZE; j++) {
                        if (field[i][j].state == CROSS) {
                        fprintf(file, "%d ", 1);
                        }
                        else if (field[i][j].state == CIRCLE) {
                            fprintf(file, "%d ", 2);
                    }
                    else {
                        fprintf(file, "%d ", 0);
                    }
                }
            }
            fclose(file);


            glClear(GL_COLOR_BUFFER_BIT);
            glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
            glTranslatef(VIEW_SIZE*0.5, VIEW_SIZE*0.5, 0);
            glScalef(VIEW_SIZE/2.0, VIEW_SIZE/2.0, 1);
            glutDisplayFunc(menuScreen);
            menuScreen();
           }
      }

      void mouse(int button, int state, int mousex, int mousey){
          int x, y;
          if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
           if(mousex >= 316 && mousex <= 385 && mousey >= 282 &&
mousey <= 385 && flagPause == 1 && flagSettings == 0 && flagWin ==
0 && strlen(fieldSize) != 0) {
                flagPause = 0;
                init();
```

```
                if(flagStart == 0) {
                     start_game();
                     flagStart = 1;
                }
                t1 = clock();
                glClear(GL_COLOR_BUFFER_BIT);
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
                glScalef(2.0/VIEW_SIZE, 2.0/VIEW_SIZE, 1);
                glTranslatef(-VIEW_SIZE*0.5, -VIEW_SIZE*0.5, 0);

                glutDisplayFunc(display);

                display();
           }
           else if(flagStart == 1 && flagPause == 0) {
                x = (mousex / ceil(WINDOW_SIZE / VIEW_SIZE)) +
current_x;
                y = ((WINDOW_SIZE - mousey) / ceil(WINDOW_SIZE /
VIEW_SIZE)) + current_y;
                if(field[x][y].state == FREE) {
                     field[x][y].state = player_move;
                     current move = AI;
                     amount_moves++;
                     free_cell--;
                     display();
                     game();
                }
           }
           else if(mousex >= 303 && mousex <= 399 && mousey >= 413
&& mousey <= 450 && flagPause == 1 && flagSettings == 0 && flagWin
== 0) {
                flagPause = 0;
                if(flagStart == 0) {
                     FILE *fp = fopen("data.txt", "r");
                     fscanf(fp, "%d", &FIELD_SIZE);
```

```
                    load();
                    current_move = PLAYER;
                    if (player_move == CROSS) ai_move = CIRCLE;
                    else ai_move = CROSS;
                    flagStart = 1;
                    game();
               }
               glClear(GL_COLOR_BUFFER_BIT);
               glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
               glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
               glScalef(2.0/VIEW_SIZE, 2.0/VIEW_SIZE, 1);
               glTranslatef(-VIEW_SIZE*0.5, -VIEW_SIZE*0.5, 0);

               glutDisplayFunc(display);

               display();
          }
          else if(mousex >= 276 && mousex <= 424 && mousey >= 559
&& mousey <= 584 && flagSettings == 0 && flagPause == 1) {
               flagSettings = 1;
               glClear(GL_COLOR_BUFFER_BIT);
               glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
               glutDisplayFunc(settingsScreen);
               settingsScreen();
          }

          else if(mousex >= 67 && mousex <= 171 && mousey >= 524
&& mousey <= 550 && flagSettings == 1 && flagPause == 1) {
               FIELD_SIZE = atoi(fieldSize);
               WIN_LEN = atoi(winLen);
               LVL = atoi(diffLevel);
               choice = atoi(whoStart);
```

```
                if(FIELD_SIZE >= 3 && WIN_LEN >= 3 ||
strlen(winLen) == 0 && strlen(fieldSize) == 0 && strlen(whoStart)
== 0 && strlen(diffLevel) == 0) {
                    flagSettings = 0;
                    glClear(GL_COLOR_BUFFER_BIT);
                    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                    glutDisplayFunc(menuScreen);
                    menuScreen();
                }

            }

            else if(mousex >= 292 && mousex <= 396 && mousey >= 631
&& mousey <= 655 && flagAbout == 0) {
                flagAbout = 1;
                glClear(GL_COLOR_BUFFER_BIT);
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                glutDisplayFunc(aboutScreen);
                aboutScreen();
            }

            else if(mousex >= 96 && mousex <= 179 && mousey >= 630
&& mousey <= 652 && flagAbout == 1) {
                flagAbout = 0;
                glClear(GL_COLOR_BUFFER_BIT);
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                glutDisplayFunc(menuScreen);
                menuScreen();
            }

            else if(mousex >= 110 && mousex <= 225 && mousey >= 587
&& mousey <= 612 && flagWin == 1) {
                // back button
                flagWin = 0;
                flagPause = 1;
                glClear(GL_COLOR_BUFFER_BIT);
```

```
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                glScalef(VIEW_SIZE/2.0, VIEW_SIZE/2.0, 1);
                                    glScalef(1/(VIEW_SIZE*0.014),
1/(-VIEW_SIZE*0.014), 1);

                glutDisplayFunc(menuScreen);
                menuScreen();
            }

            else if(mousex >= 279 && mousex <= 423 && mousey >= 489
&& mousey <= 514 && flagRecords == 0) {
                flagRecords = 1;
                glClear(GL_COLOR_BUFFER_BIT);
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

                glutDisplayFunc(recordsScreen);
                recordsScreen();
            }

            else if(mousex >= 95 && mousex <= 184 && mousey >= 630
&& mousey <= 649 && flagRecords == 1) {
                flagRecords = 0;
                glClear(GL_COLOR_BUFFER_BIT);
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

                glutDisplayFunc(menuScreen);
                menuScreen();
            }

        }
    }

    int main(int argc, char *argv[]) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(WINDOW_SIZE, WINDOW_SIZE);
```

```
glutInitWindowPosition((glutGet(GLUT_SCREEN_WIDTH)-WINDOW_SIZE)   /
2, (glutGet(GLUT_SCREEN_HEIGHT)-WINDOW_SIZE) / 2);
          glutCreateWindow("TicTacToe");

          glutDisplayFunc(menuScreen);
          glutMouseFunc(mouse);
          glutSpecialFunc(keyboard1);
          glutKeyboardFunc(keyboard2);

          glutMainLoop();
          return 0;
      }
```