

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 "1942"»

Выполнил:

студент группы 25BBB1

Цеплов Е. В.

Принял:

к.т.н., доцент Токарев А. Н.

к.т.н., доцент Генералова А. А.

001

27.05.26

*Генералова*

Пенза 2026

## “УТВЕРЖДАЮ”

М.А. Митрохин

« \_\_\_\_\_ » 20\_\_ г.

Студенту Ченявс Егору Владимировичу Группа 25ВВВ1

Игра "1942"

Игра "1942"  
Игра управляет самолетами, прямыми над картой и уничтожает самолеты противника.  
Сохранение и загрузка из файлов таблицы рекордов.

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 " 18 " 02 2026

Дата защиты проекта " " "

Руководитель Тюкарев А.М.

Задание получил " 18 " 02 2026 г.

Студент Ценев Е.В.

## Содержание

|                                                 |    |
|-------------------------------------------------|----|
| Введение .....                                  | 5  |
| Постановка задачи.....                          | 6  |
| Выбор решения.....                              | 7  |
| Описание разработки программы .....             | 8  |
| Отладка и тестирование .....                    | 9  |
| Описание программы .....                        | 9  |
| Руководство пользователя .....                  | 20 |
| Заключение .....                                | 25 |
| Список используемых источников.....             | 26 |
| Приложение А. Листинги программы .....          | 27 |
| Приложение В. Результаты работы программы ..... | 71 |

## Введение

Компьютерные игры являются одним из наиболее наглядных и эффективных способов изучения программирования, так как объединяют работу с данными, обработку ввода пользователя, построение интерфейса и организацию непрерывного игрового цикла. Даже относительно простая игра требует тщательно продуманной архитектуры: необходимо управлять состоянием игровых объектов, реагировать на действия игрока, просчитывать физику движения, обрабатывать столкновения и своевременно обновлять визуальный вывод.

В данной курсовой работе рассматривается разработка графической игры «1942» на языке C с использованием библиотеки `raylib`. Игра является современной интерпретацией классического аркадного скролл-шутера: игрок управляет самолётом, уничтожает волны врагов различных типов и в финале сражается с боссом. Проект выбран как пример программы сложной структуры: в нём присутствует главный игровой цикл, система анимации спрайтов, физика движения, многотипная система врагов, обработка столкновений, ведение счёта и таблица рекордов.

Разработанная программа использует библиотеку `raylib` для отрисовки графики, воспроизведения звука и обработки пользовательского ввода. Исходный код разбит на несколько модулей, каждый из которых отвечает за строго определённую область функциональности, что соответствует принципу нисходящего программирования.

Цель курсовой работы — разработать графическую игру «1942», реализующую управление самолётом игрока, несколько типов врагов с различным поведением, систему пуль, обнаружение столкновений, анимацию спрайтов, скроллируемый фон, интерфейс HUD, главное меню и таблицу рекордов с сохранением в файл.

## Постановка задачи

Необходимо разработать графическую игру «1942» — вертикальный скролл-шутер, в котором игрок управляет самолётом и уничтожает волны вражеских самолётов. Программа должна обеспечивать плавное движение игрока, систему стрельбы, появление врагов по заранее заданному сценарию уровня, обработку столкновений и ведение очков.

Игровое поле должно иметь фиксированный размер  $768 \times 1024$  пикселей с вертикально скроллируемым фоном из трёх сменяющихся текстур, создающих эффект бесконечного движения. Игрок управляет самолётом, который может двигаться по всему экрану и вести огонь из двух стволов одновременно.

В игре должны присутствовать три типа врагов:

- Камикадзе — летит вниз по синусоидальной траектории, не стреляет, уничтожается двумя попаданиями;
- Файтер — занимает позицию и ведёт огонь прямо вниз, уничтожается четырьмя попаданиями;
- Босс — появляется в финале уровня, движется по горизонтали и стреляет веером три пули, имеет 80 единиц здоровья.

Программа должна реализовывать систему очков: за уничтожение камикадзе начисляется 100 очков, за файтера — 200, за босса — 5000. Лучшие результаты с именем игрока и отметкой победы/поражения должны сохраняться в файл и отображаться в таблице рекордов TOP-10.

Необходимо реализовать главное меню с пунктами «Play», «Records» и «Exit», экран окончания игры с вводом имени и отображением таблицы, а также HUD с текущим счётом, полосками жизней и таймером.

Программа должна быть разбита на модули, включать в себя сложные типы данных, а так же ассемблерную вставку.

## Выбор решения

Для решения задачи выбран язык программирования C, так как он позволяет явно управлять памятью, структурами данных. В качестве графической библиотеки использована `raylib` — кроссплатформенная библиотека для разработки игр на C, предоставляющая функции для работы с окном, текстурами, звуком, клавиатурой и таймером без необходимости устанавливать тяжёлые фреймворки.

В функции `Anim_Frame` используется встроенная ассемблерная вставка для вычисления смещения кадра спрайта выполняется через регистр `eax` процессора с помощью инструкции `imul`. К преимуществам использования ассемблера относятся возможность полного контроля над регистровой моделью процессора, минимальные накладные расходы при выполнении операций.

В качестве среды разработки была выбрана Microsoft Visual Studio 2022, которая является одной из наиболее функциональных IDE для разработки приложений. Среда поддерживает работу с проектами на C, обеспечивает удобную навигацию по исходному коду и встроенные средства отладки.

Выбранная совокупность языков программирования и среды разработки полностью соответствует поставленным задачам, обеспечивает высокую производительность приложения, позволяет продемонстрировать как базовые, так и углублённые навыки программирования, а также позволяет продемонстрировать широкий спектр навыков программирования, включая работу с системными ресурсами, графикой, структурами данных и низкоуровневыми механизмами выполнения программ.

## Описание разработки программы

Разработка программы выполнялась поэтапно в соответствии с нисходящим подходом. Сначала был создан заголовочный файл `game.h` со всеми константами, структурами и прототипами функций. Затем реализована точка входа в `main.c` с главным циклом. В последнюю очередь добавлялись отдельные игровые подсистемы.

Порядок реализации модулей:

- `game.h` — общие объявления типов, констант и прототипов;
- `anim.c` — система покадровой анимации спрайтов с ассемблерной вставкой;
- `background.c` — бесконечно скроллируемый фон;
- `player.c` — управление самолётом игрока;
- `bullets.c` — система пуль игрока и врагов;
- `enemies.c` — поведение трёх типов врагов;
- `level.c` — сценарий появления волн;
- `collision.c` — обнаружение и обработка столкновений;
- `hud.c` — игровой интерфейс и экран окончания;
- `menu.c` — главное меню;
- `scoreboard.c` — таблица рекордов;
- `render_utils.c` — утилита теневых эффектов;
- `main.c` — точка входа, главный цикл.

По порядку реализации модулей видно, что использовался нисходящий подход проектирования.

## **Отладка и тестирование**

Отладка программы проводилась после реализации каждого основного модуля в среде Microsoft Visual Studio 2022, которая обладает всеми средствами, необходимыми при разработке и отладке программы. Тестирование проводилось во время разработки и также после её завершения.

Сначала проверялась корректность запуска приложения, загрузки текстур и звуков, а также переход из главного меню в игровой режим.

Система пуль тестировалась на корректность интервала стрельбы. Поведение каждого типа врагов проверялось отдельно. Обнаружение столкновений проверялось для всех трёх случаев. Отдельно тестировалось начисление очков, таймер неуязвимости и условие победы при уничтожении босса.

Таблица рекордов проверялась на корректное сохранение и загрузку, сортировку по убыванию очков и ограничение в 10 записей.

## **Описание программы**

Программа «1942» представляет собой графическое оконное приложение. При запуске производится инициализация графического окна разрешением 768×1024 пикселей, загрузка всех текстур (фоны, спрайты, иконки) и звуковых файлов. После этого программа проверяет наличие файла таблицы рекордов scores.txt. Если файл отсутствует, он создаётся автоматически при первом сохранении результата. Затем отображается главное меню, где пользователь может выбрать один из трёх пунктов: «Play» (начать игру), «Records» (просмотр таблицы рекордов) или «Exit» (выход). Во время игры пользователь управляет самолётом с помощью клавиатуры. На каждом кадре (60 раз в секунду) обновляются положения игрока, пуль и врагов, проверяются столкновения, отображается игровая статистика (счёт,

жизни, таймер). При достижении последнего события уровня (появление босса и его уничтожение) или при потере всех жизней игра завершается, открывается экран окончания с предложением ввести имя для сохранения результата в таблицу рекордов.

В таблице 1 представлено описание взаимодействия пользователя с программой.

**Таблица 1 - Взаимодействие с программой**

| <b>Клавиши /<br/>устройство ввода</b> | <b>Действие<br/>пользователя</b> | <b>Действие программы</b>                                 |
|---------------------------------------|----------------------------------|-----------------------------------------------------------|
| W / Стрелка<br>вверх                  | Удерживание<br>клавиши           | Движение самолёта<br>вверх, фон ускоряется                |
| S / Стрелка<br>вниз                   | Удерживание<br>клавиши           | Движение самолёта<br>вниз                                 |
| A / Стрелка<br>влево                  | Удерживание<br>клавиши           | Движение влево, крен<br>самолёта влево до $-25^{\circ}$   |
| D / Стрелка<br>вправо                 | Удерживание<br>клавиши           | Движение вправо, крен<br>самолёта вправо до $+25^{\circ}$ |
| Пробел                                | Нажатие                          | Выстрел из двух<br>стволов (интервал 0,15 с)              |
| W / S в меню                          | Нажатие                          | Навигация по пунктам<br>меню                              |
| Enter / Пробел<br>в меню              | Нажатие                          | Выбор пункта меню                                         |
| R (экран<br>окончания)                | Нажатие после<br>сохранения      | Перезапуск игровой<br>сессии                              |
| BackSpace<br>(экран окончания)        | Нажатие после<br>сохранения      | Возврат в главное<br>меню                                 |

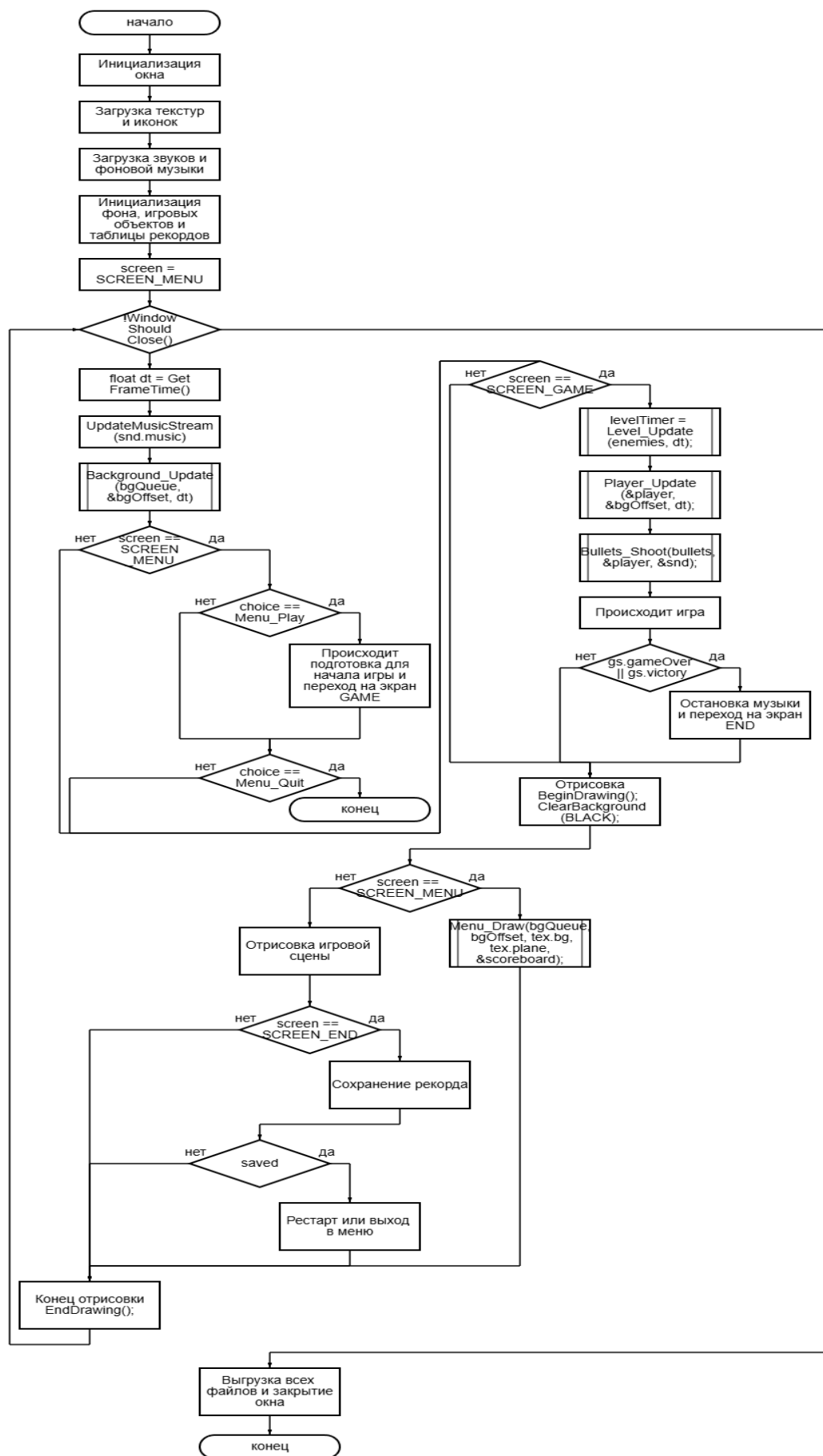


Рисунок 1 –Функция main

Функция `main()` является точкой входа. Она инициализирует окно, загружает ресурсы, создаёт игровые объекты (массивы пуль, врагов, структуры игрока и состояния игры). Затем входит в бесконечный цикл, работающий до закрытия окна. На каждой итерации вычисляется время `dt`, обновляется музыкальный поток, а в зависимости от текущего экрана (`SCREEN_MENU`, `SCREEN_GAME`, `SCREEN_END`) вызываются соответствующие функции обновления и отрисовки. После отрисовки кадр выводится на экран. При закрытии окна ресурсы выгружаются.

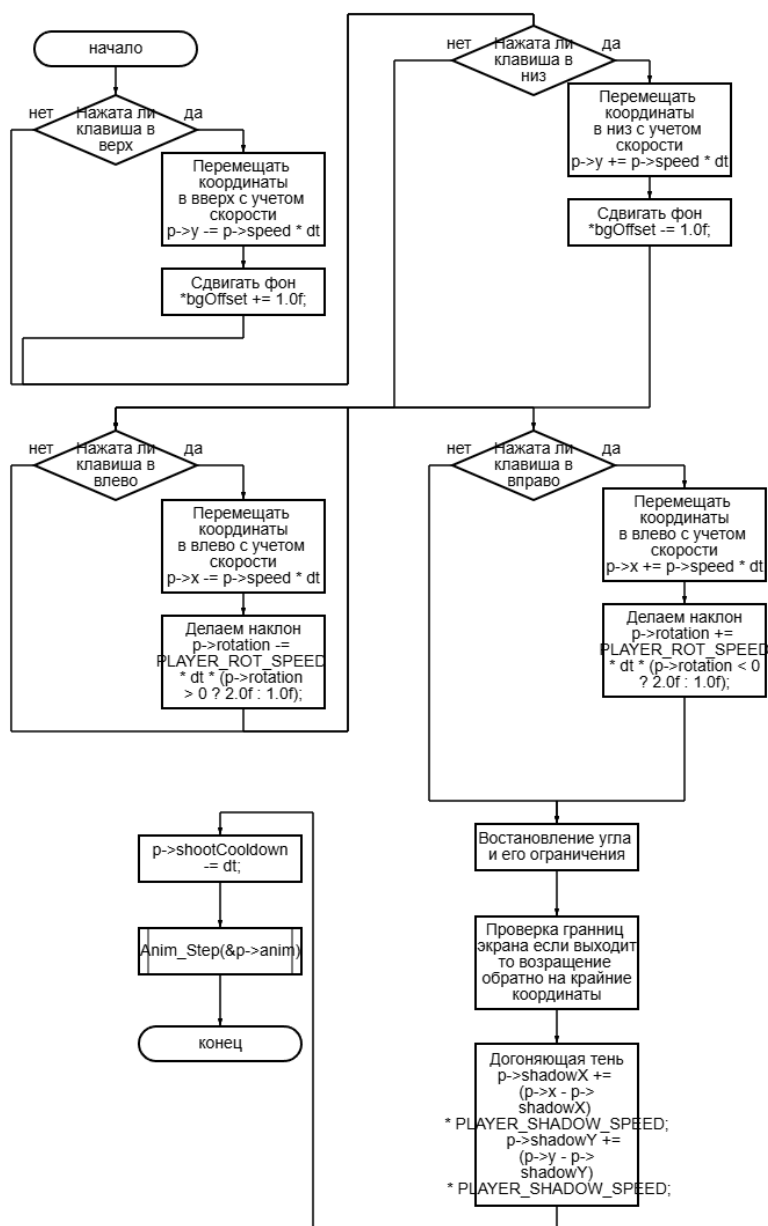


Рисунок 2—Функция `Player_Update`

Функция `Player_Update()` отвечает за обработку управления самолётом игрока, обновление его положения, угла крена и тени, а также за перезарядку оружия и анимацию. Она вызывается на каждом кадре главного игрового цикла (60 раз в секунду) и получает текущее состояние клавиатуры через функции `IsKeyDown()`.

Сначала функция проверяет нажатия клавиш движения (W/A/S/D или стрелки). При движении вверх или вниз изменяется координата у игрока и незначительно смещается фон. При движении влево или вправо изменяется координата x и плавно корректируется угол крена (rotation) самолёта с максимальным отклонением  $\pm 25^\circ$ . Если клавиши движения не нажаты, угол крена постепенно возвращается к нулю. После обработки ввода выполняются ограничения положения игрока в пределах экрана (с учётом размеров спрайта).

Затем обновляется положение тени — она движется с задержкой относительно самого самолёта, создавая эффект инерции. Уменьшается таймер перезарядки оружия (`shootCooldown`). В конце вызывается `Anim_Step()`, которая переключает кадры анимации через заданные интервалы времени.

Таким образом, `Player_Update()` полностью реализует всю динамику управления игроком: плавное перемещение, крен, ограничения, тень и анимацию.

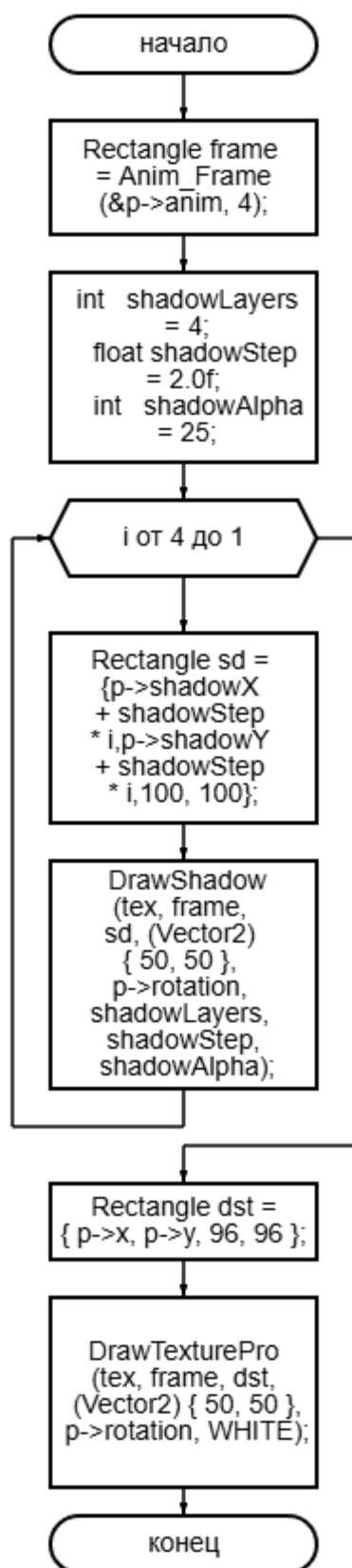


Рисунок 3—Функция Player\_Draw

Функция `Player_Draw()` отвечает за визуализацию самолёта игрока на экране. Она вызывается после завершения всех логических обновлений (движения, столкновений) и получает текущую текстуру самолёта, а также данные о положении, угле крена и текущем кадре анимации.

В начале функции с помощью `Anim_Frame()` вычисляется прямоугольник источника (`src`) в спрайт-листе, соответствующий текущему анимационному кадру (ширина кадра 59 пикселей, высота 64). Затем создаётся прямоугольник назначения (`dst`) с размерами 96×96 пикселей, расположенный в координатах самолёта (`x`, `y`).

Далее в цикле отрисовывается многослойная тень: для каждого из четырёх слоёв вычисляется смещение на  $i * \text{step}$  пикселей вниз и вправо, прозрачность уменьшается с каждым слоём. Функция `DrawShadow()` (из модуля `render_utils.c`) рисует каждый слой с соответствующим уровнем прозрачности чёрного цвета.

После тени отрисовывается сам самолёт: вызывается `DrawTexturePro()` с текстурой, вычисленным фрагментом спрайт-листа, целевым прямоугольником, точкой вращения (центр спрайта, 50×50), углом `rotation` (крен) и белым цветом (`WHITE`). Благодаря этому самолёт отображается с плавным поворотом при движении влево/вправо и с анимированными кадрами (пропеллер, эффект полёта).

Таким образом, функция `Player_Draw()` полностью реализует визуальное представление игрока: анимированный спрайт, объёмную тень и динамический угол крена.

Таким образом представленные функции представляют собой базу игры отрисовка самолета и его управление.

Основные модули программы:

### **Модуль anim.c**

Модуль содержит функции Anim\_Step и Anim\_Frame. Функция Anim\_Step уменьшает оставшееся время текущего кадра на dt и при необходимости переключает на следующий кадр по кольцу. Функция Anim\_Frame вычисляет прямоугольник нужного кадра в спрайт-листе. Для вычисления смещения используется встроенная ассемблерная вставка: индекс кадра умножается на ширину кадра (59 пикселей) через регистр eax процессора с помощью инструкции imul, что является демонстрацией низкоуровневых операций в проекте.

### **Модуль background.c**

Модуль реализует бесконечно скроллируемый фон из очереди сегментов. Функция Background\_Init заполняет очередь случайными индексами трёх доступных текстур фона. Background\_Update каждый кадр увеличивает смещение на 150 пикселей в секунду и при достижении границы сдвигает очередь, добавляя новый случайный сегмент. Background\_Draw рисует все сегменты с учётом текущего смещения.

### **Модуль player.c**

Модуль управляет самолётом игрока. Player\_Init устанавливает начальное положение в нижней части экрана. Player\_Update обрабатывает нажатия клавиш W/A/S/D и стрелок, изменяет положение и плавно меняет угол крена при движении по горизонтали (максимум  $\pm 25$  градусов). Тень самолёта движется с задержкой, создавая эффект инерции. Player\_Draw рисует многослойную тень и текущий кадр анимации с поворотом.

### **Модуль bullets.c**

Модуль управляет пулями игрока и врагов через два массива фиксированного размера. Bullets\_Shoot проверяет нажатие пробела и интервал перезарядки, после чего активирует две пули — от левого и правого ствола. Bullets\_DrawEnemy вычисляет угол поворота пули врага по вектору её скорости для правильной ориентации спрайта.

### **Модуль enemies.c**

Модуль реализует поведение всех трёх типов врагов. `Enemy_Spawn` ищет свободный слот в массиве и инициализирует врага параметрами своего типа. `Enemies_Update` обновляет каждого активного врага: камикадзе летит вниз по синусоиде; файтер движется до целевой Y-координаты, затем покачивается и стреляет; босс движется по горизонтали и выпускает трёхпулевой веер. `Enemies_Draw` рисует тени, спрайты, полосы HP и HP-бар босса.

### **Модуль collision.c**

Модуль содержит три функции проверки столкновений. `Collision_BulletsVsEnemies` проверяет попадание пуль игрока в врагов через попарное сравнение прямоугольников. `Collision_EnemyBulletsVsPlayer` проверяет попадание пуль врагов в хитбокс игрока с учётом таймера неуязвимости. `Collision_EnemiesVsPlayer` обрабатывает прямой таран врагов.

### **Модуль level.c**

Модуль содержит массив `gLevel` из структур `LevelEvent`, описывающий сценарий уровня продолжительностью около 5 минут. Уровень разбит на волны: первые камикадзе появляются на 3-й секунде, файтеры — на 15-й, финальный босс — на 295-й секунде. Функция `Level_Update` прибавляет `dt` к таймеру и последовательно спавнит врагов, время которых уже наступило.

### **Модуль hud.c**

Модуль отвечает за игровой интерфейс. `HUD_Draw` выводит текущий счёт, пять иконок жизней и обратный таймер. `HUD_DrawEndScreen` управляет экраном окончания игры: принимает ввод имени игрока с клавиатуры, отображает поле ввода с мигающим курсором, передаёт результат в таблицу рекордов и показывает занятое место.

### **Модуль menu.c**

Модуль реализует главное меню с тремя пунктами: «Play», «Records» и «Exit». `Menu_Update` обрабатывает навигацию клавишами и возвращает выбранный пункт при нажатии Enter или пробела. `Menu_Draw` рисует анимированный фон, заголовок «1942» с золотым цветом и тенью, три

кнопки с подсветкой, а также мини-таблицу рекордов при выборе пункта «Records».

### Модуль scoreboard.c

Модуль управляет таблицей рекордов. Scoreboard\_Load считывает структуру из бинарного файла scores.txt. Scoreboard\_Submit вставляет новую запись на правильную позицию по убыванию очков. Scoreboard\_Draw рисует панель с таблицей, подсвечивая только что добавленную запись.

### Модуль render\_utils.c

Утилитная функция DrawShadow послойно рисует смещённые полупрозрачные копии текстуры, имитируя объёмную тень. Количество слоёв, шаг смещения и прозрачность передаются как параметры.

Программа требует наличия библиотеки `raylib` и файлов ресурсов (текстуры PNG, звуки MP3) в рабочей директории. Таблица рекордов хранится в бинарном файле `scores.txt` и создаётся автоматически при первом завершении партии.

Далее на рисунке 4 представлена схема взаимодействия программы.

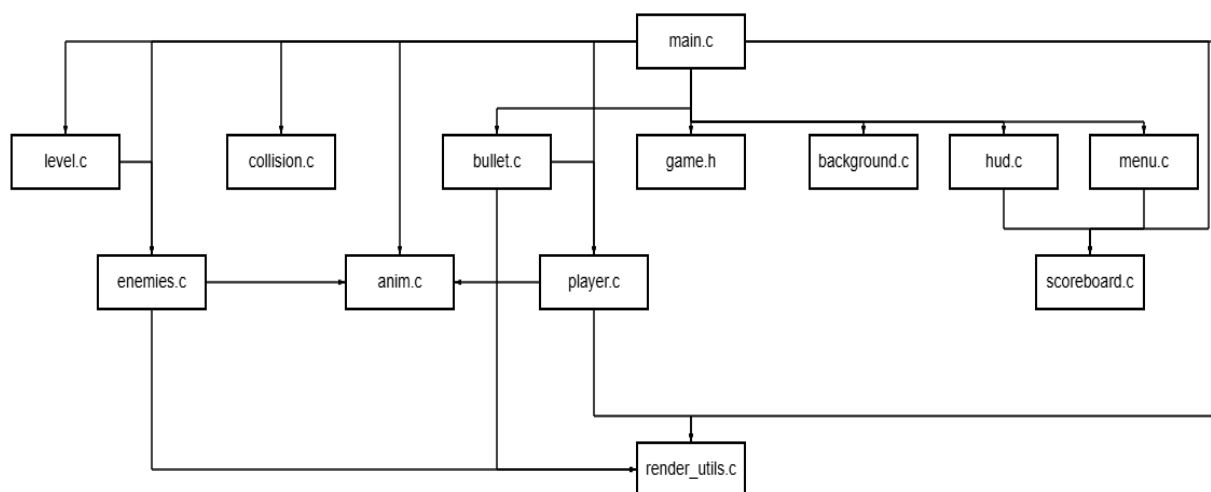


Рисунок 4—схема взаимодействия программы

Ниже, на рисунке 5 представлена схема данных.



Рисунок 5—схема данных

Ниже приведена иерархическая схема программы.

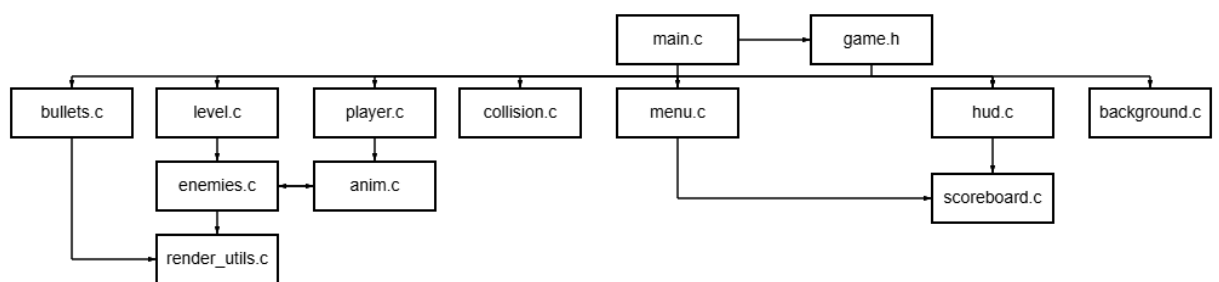


Рисунок 6—иерархическая схема программы

## Руководство пользователя

Для запуска программы необходимо открыть исполняемый файл игры. После запуска на экране появляется главное меню (рисунок 1) с тремя пунктами: «Play», «Records» и «Exit».

Навигация по меню выполняется клавишами W/S или стрелками вверх/вниз. Выбор подтверждается клавишей Enter или пробелом. Выбранный пункт подсвечивается жёлтым цветом и обозначается мигающим маркером «>». При выборе пункта «Records» в нижней части экрана появляется мини-таблица рекордов.

Для начала игры необходимо выбрать пункт «Play». Самолёт игрока расположен в нижней части экрана. Управление в игре описано в таблице 1.

Цель игры — уничтожать вражеские самолёты и набирать очки. За камикадзе начисляется 100 очков, за файтера — 200 очков. Финальный босс (рисунок 2) появляется примерно через 5 минут и приносит 5000 очков при уничтожении. Уничтожение босса означает победу.

В верхней части экрана (рисунок 3) отображается HUD: текущий счёт слева, пять иконок жизней в виде сердечек и обратный таймер справа. Игрок начинает с 5 жизнями. При получении урона экран мигает красным, а игрок на 1,5 секунды становится неуязвимым. При потере всех жизней наступает поражение.

При победе или поражении открывается экран окончания игры (рисунок 4). Необходимо ввести имя и нажать Enter для сохранения результата. После сохранения отображается таблица рекордов TOP-10 с подсвеченной новой записью. Для повторной игры нажмите R, для возврата в меню — Escape.

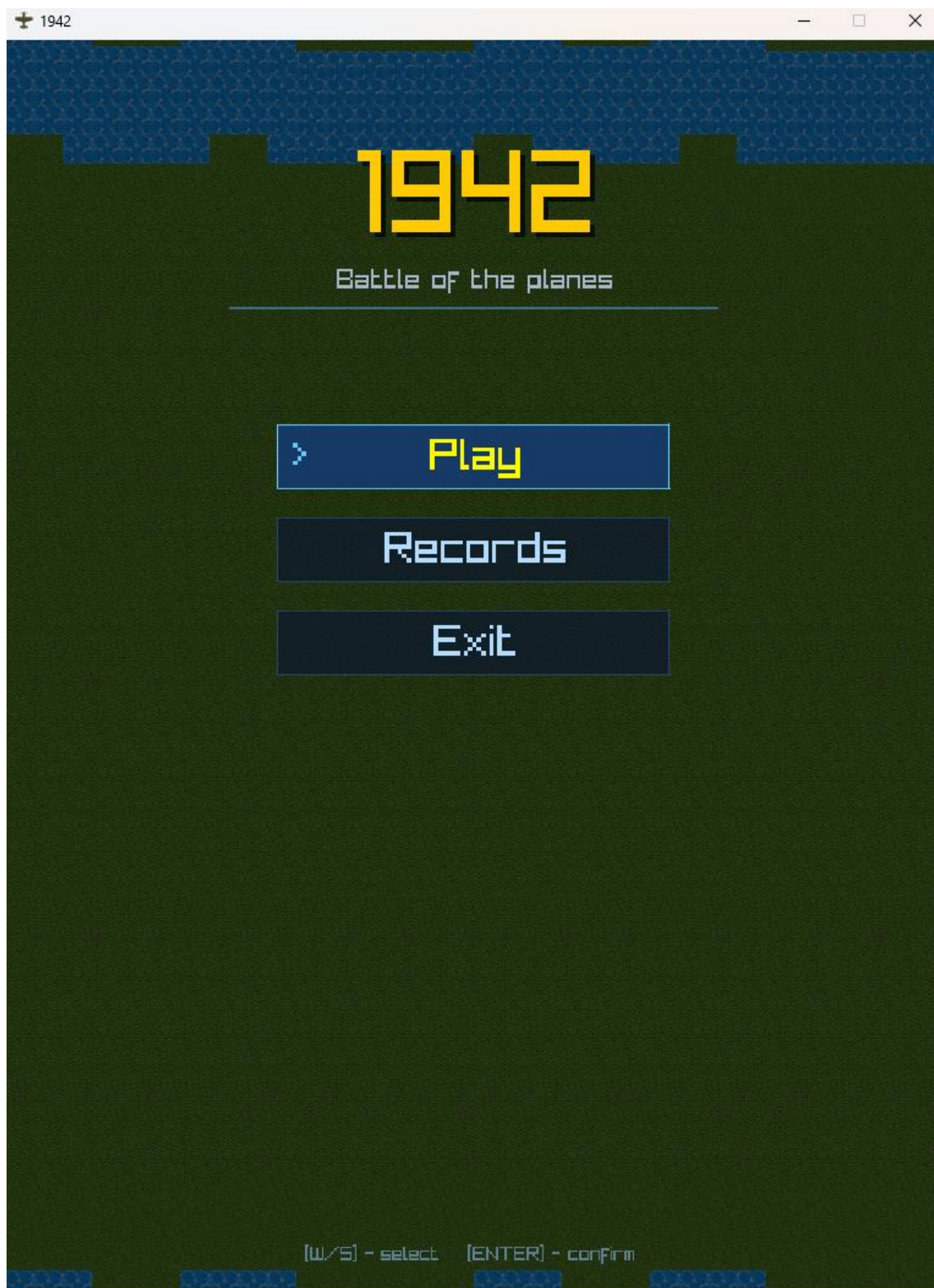


Рисунок 7 — Главное меню

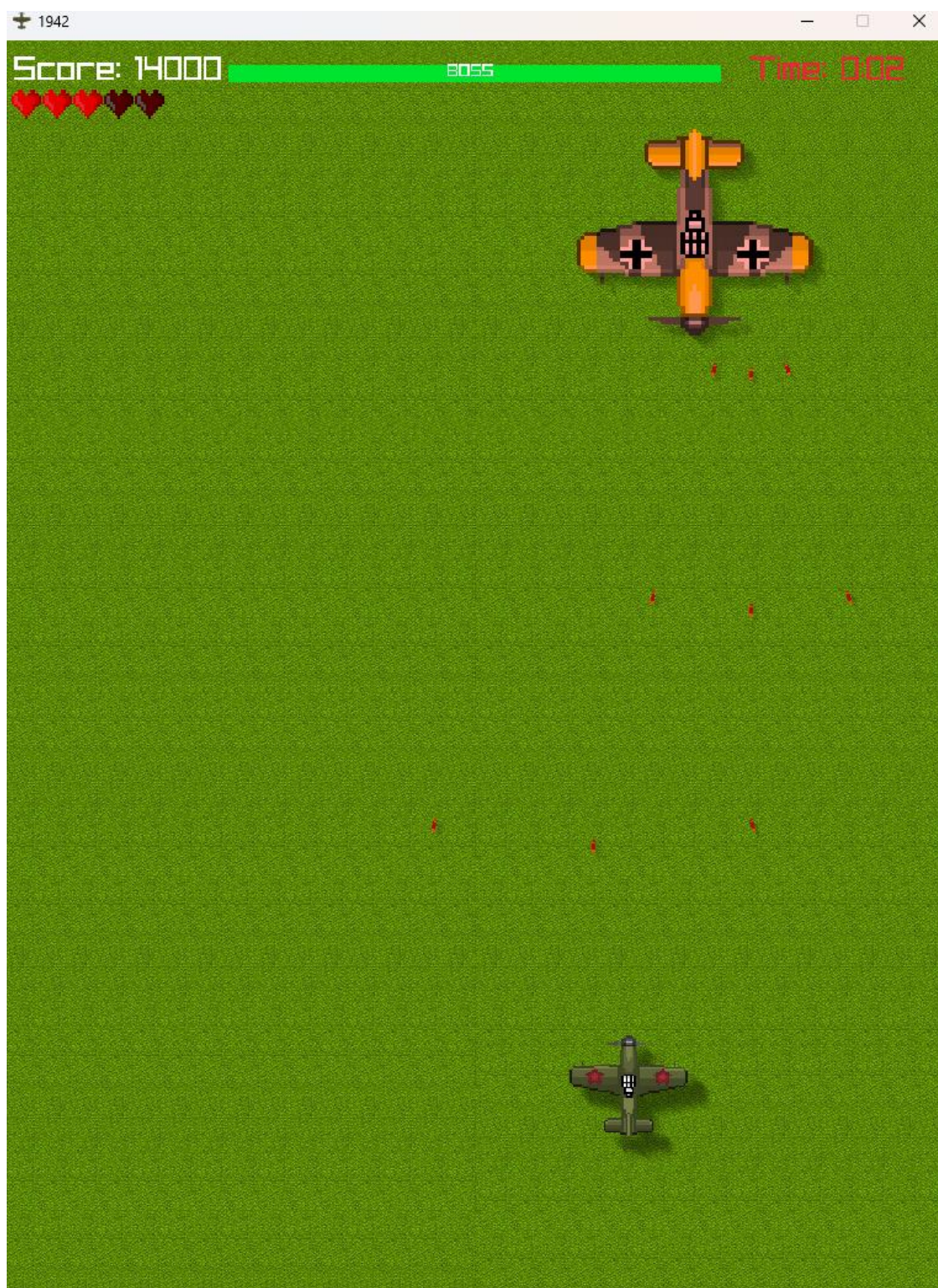


Рисунок 8 — Финальный босс

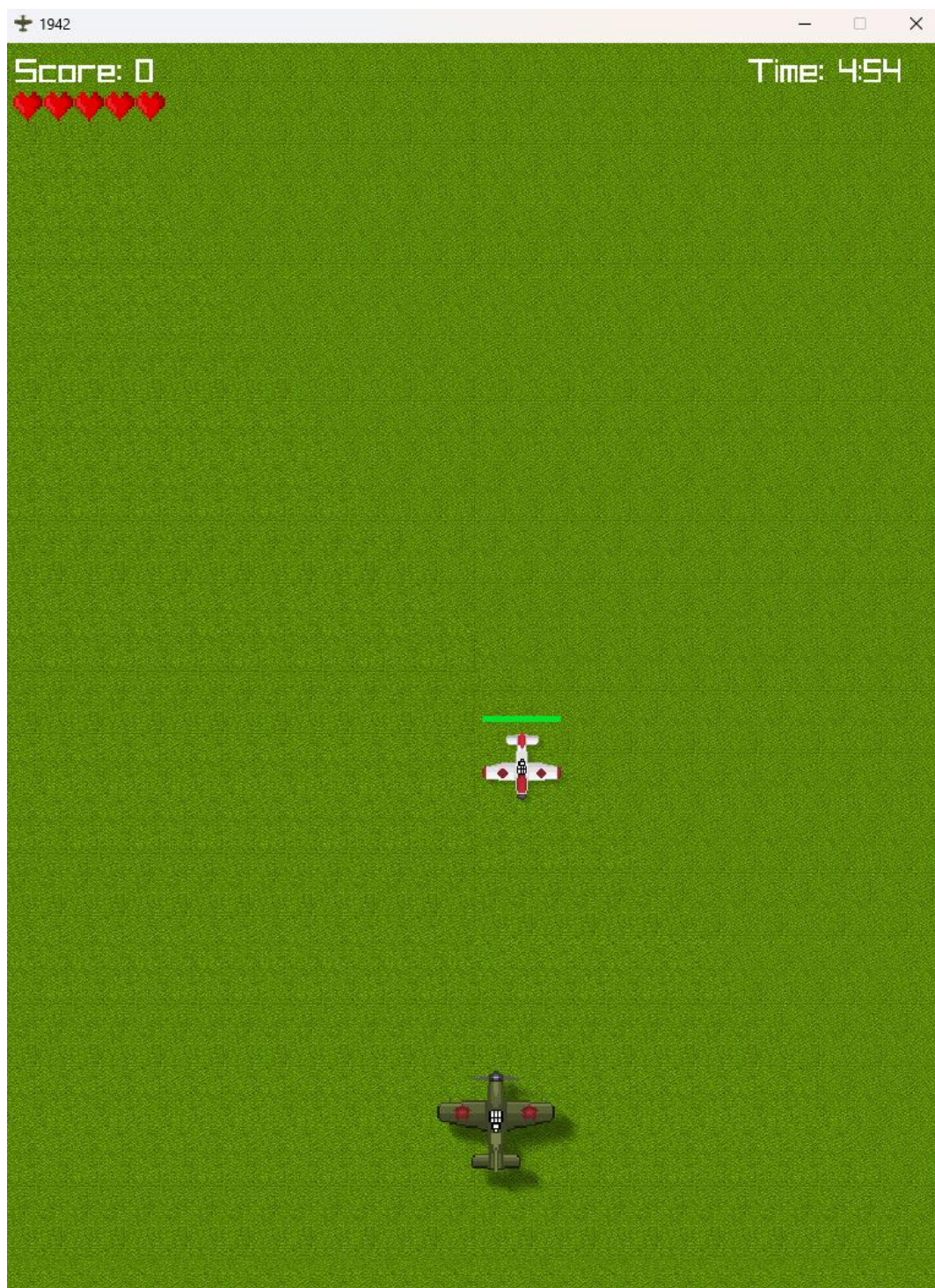


Рисунок 9 — Игровой процесс (HUD)



Рисунок 10 — Экран окончания игры

## Заключение

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ на языке C с использованием библиотеки `raylib`. Освоена работа с графикой, анимацией, звуком и файловой системой. Были получены практические навыки создания игрового цикла, обработки пользовательского ввода с клавиатуры, реализации физики движения и системы столкновений. Также были закреплены навыки отладки и тестирования программ в среде Microsoft Visual Studio 2022 на языках C и Assembler.

В процессе разработки были применены принципы нисходящего модульного программирования. Исходный код разделён на 13 файлов, каждый из которых отвечает за конкретную часть программы. В проекте использована встроенная ассемблерная вставка в функции `Anim_Frame` для вычисления смещения кадра спрайта через регистр `eax` процессора.

Разработанная программа соответствует поставленной задаче. Полученный проект может быть расширен: например, можно добавить несколько уровней, дополнительные типы оружия, систему бонусов или поддержку геймпада.

## Список используемых источников

1. Керниган Б., Ритчи Д. Язык программирования С. — 2-е изд. — СПб.: Вильямс, 2019. — 304 с.
2. Павловская Т. А. С/С++. Программирование на языке высокого уровня. — СПб.: Питер, 2021. — 432 с.
3. Ирвин К. Язык ассемблера для процессоров Intel. — 6-е изд. — М.: Вильямс, 2013. — 912 с.
4. raylib — A simple and easy-to-use library to enjoy videogames programming [Электронный ресурс]. — URL: <https://www.raylib.com> (дата обращения: 01.05.2026).
5. raylib cheatsheet [Электронный ресурс]. — URL: <https://www.raylib.com/cheatsheet/cheatsheet.html> (дата обращения: 01.05.2026).
6. ISO/IEC 9899:2018. Programming languages — C. — Geneva: ISO, 2018.

## Приложение А. Листинги программы

### Файл game.h

```
#include <raylib.h>
#include <stdbool.h>
#include <math.h>
#include <stdlib.h>

// КОНСТАНТЫ
#define SCREEN_W          768
#define SCREEN_H          1024
#define MAX_BULLETS       64
#define MAX_ENEMIES       32
#define MAX_ENE_BULLETS   64
#define BG_SEGMENTS       12

// ПЕРЕЧИСЛЕНИЯ
typedef enum {
    ENEMY_KAMIKAZE = 0,
    ENEMY_FIGHTER = 1,
    ENEMY_BOSS = 2,
} EnemyType;

typedef enum {
    SCREEN_MENU = 0,
    SCREEN_GAME,
    SCREEN_END,
} GameScreen;

typedef enum {
    MENU_PLAY = 0,
    MENU_SCORES,
    MENU_QUIT,
    MENU_COUNT,
} MenuItem;

// СТРУКТУРЫ
typedef struct {
    int    first, last, cur;
    float  speed, duration;
} Animation;

typedef struct {
    float x, y;
    float speedX, speedY;
```

```

        bool    active;
    } Bullet;

typedef struct {
    float      x, y;
    float      shadowX, shadowY;
    float      baseX;
    float      waveTime;
    float      hp, maxHp;
    float      speed;
    bool       active;
    float      shootTimer;
    float      shootInterval;
    Animation  anim;
    EnemyType  type;
    float      targetY;
    bool       reachedTarget;
    float      bossDir;
    float      bossTimer;
    float      deathTimer;
} Enemy;

typedef struct {
    float      time;
    float      x;
    EnemyType  type;
} LevelEvent;

typedef struct {
    int    score;
    int    playerHp;
    float  invincibleTimer;
    bool   gameOver;
    bool   victory;
    bool   bossAlive;
} GameState;

typedef struct {
    float x, y;
    float shadowX, shadowY;
    float speed;

```

```

        float rotation;
        float shootCooldown;
        Animation anim;
    } Player;

typedef struct {
    Texture2D bg[3];
    Texture2D plane;
    Texture2D enemyA;
    Texture2D enemyB;
    Texture2D bullet;
    Texture2D heart;
} Textures;

typedef struct {
    Music  music;
    Sound  shoot;
    Sound  hitEnemy;
    Sound  hitPlayer;
    Sound  gameOver;
} Sounds;

// МОДУЛЬ: animation  (anim.c)
void      Anim_Step(Animation* a);
Rectangle Anim_Frame(Animation* a, int numFrames);
// МОДУЛЬ: background  (background.c)
void Background_Init(int queue[BG_SEGMENTS]);
void Background_Update(int queue[BG_SEGMENTS], float* offset, float
dt);

void Background_Draw(int queue[BG_SEGMENTS], float offset, Texture2D
textures[3]);

// МОДУЛЬ: player  (player.c)
void Player_Init(Player* p);
void Player_Update(Player* p, float* bgOffset, float dt);
void Player_Draw(Player* p, Texture2D tex);
// МОДУЛЬ: bullets  (bullets.c)
void Bullets_Shoot(Bullet bullets[MAX_BULLETS], Player* p, Sounds*
snd);

void Bullets_UpdatePlayer(Bullet bullets[MAX_BULLETS], float dt);
void Bullets_UpdateEnemy(Bullet bullets[MAX_ENE_BULLETS], float dt);
void Bullets_DrawPlayer(Bullet bullets[MAX_BULLETS], Texture2D tex);
void Bullets_DrawEnemy(Bullet bullets[MAX_ENE_BULLETS], Texture2D
tex);

```

```

// МОДУЛЬ: enemies (enemies.c)
void Enemy_Spawn(Enemy enemies[MAX_ENEMIES], EnemyType type, float
x);

void Enemies_Update(Enemy enemies[MAX_ENEMIES],
    Bullet eneBullets[MAX_ENE_BULLETS],
    float planeX, float planeY, float dt);
void Enemies_Draw(Enemy enemies[MAX_ENEMIES],
    Texture2D texA, Texture2D texB);
// МОДУЛЬ: level (level.c)
extern LevelEvent gLevel[];
extern int      gTotalEvents;

void Level_Init(void);
float Level_Update(Enemy enemies[MAX_ENEMIES], float dt);
// МОДУЛЬ: collision (collision.c)
void Collision_BulletsVsEnemies(Bullet bullets[MAX_BULLETS],
    Enemy enemies[MAX_ENEMIES],
    GameState* gs, Sounds* snd);
void Collision_EnemyBulletsVsPlayer(Bullet
eneBullets[MAX_ENE_BULLETS],
    Player* p, GameState* gs, Sounds* snd);
void Collision_EnemiesVsPlayer(Enemy enemies[MAX_ENEMIES],
    Player* p, GameState* gs, Sounds* snd);
// МОДУЛЬ: scoreboard (scoreboard.c)
#define SCOREBOARD_MAX    10
#define SCOREBOARD_FILE   "scores.txt"
#define PLAYER_NAME_LEN   16

typedef struct {
    char name[PLAYER_NAME_LEN];
    int  score;
    bool victory;
} ScoreEntry;

typedef struct {
    ScoreEntry entries[SCOREBOARD_MAX];
    int        count;
} Scoreboard;

void Scoreboard_Load(Scoreboard* sb);
void Scoreboard_Save(const Scoreboard* sb);

```

```

    int Scoreboard_Submit(Scoreboard* sb, const char* name, int score,
bool victory);
    void Scoreboard_Draw(const Scoreboard* sb, int highlightIdx);
    // МОДУЛЬ: hud (hud.c)
    void HUD_Draw(GameState* gs, Texture2D heartTex, float levelTimer);
    bool HUD_DrawEndScreen(GameState* gs, Scoreboard* sb);
    void HUD_ResetNameEntry(void);
    // УТИЛИТА: тень (render_utils.c)
    void DrawShadow(Texture2D tex, Rectangle src, Rectangle dst,
        Vector2 origin, float rotation,
        int layers, float step, int alpha);
    int Menu_Update(void);
    void Menu_Draw(int bgQueue[BG_SEGMENTS], float bgOffset, Texture2D
bgTex[3], Texture2D planeTex, const Scoreboard* sb);

```

## Файл main.c

```
#include "game.h"

//Сбросигровогосостояния
static void ResetGame(Player* player, Bullet bullets[], Bullet
eneBullets[],
    Enemy enemies[], GameState* gs, Sounds* snd, float* levelTimer)
{
    Player_Init(player);

    for (int i = 0; i < MAX_BULLETS; i++) bullets[i].active = false;
    for (int i = 0; i < MAX_ENE_BULLETS; i++) eneBullets[i].active =
false;

    for (int i = 0; i < MAX_ENEMIES; i++) enemies[i].active = false;

    *gs = (GameState){
        .score = 0,
        .playerHp = 5,
        .invincibleTimer = 0.0f,
        .gameOver = false,
        .victory = false,
        .bossAlive = false,
    };

    *levelTimer = 0.0f;
    Level_Init();
    HUD_ResetNameEntry();
    PlayMusicStream(snd->music);
}

int main(void) {
    InitWindow(SCREEN_W, SCREEN_H, "1942");
    SetTargetFPS(60);

    //Текстуры
    Textures tex = {
        .bg = { LoadTexture("background.png"),
                LoadTexture("background2.png"),
```

```

        LoadTexture("background3.png") },
        .plane = LoadTexture("plane.png"),
        .enemyA = LoadTexture("enemy_A.png"),
        .enemyB = LoadTexture("enemy_B.png"),
        .bullet = LoadTexture("bullet.png"),
        .heart = LoadTexture("Heart.png"),
};

Image icon = LoadImage("Icon.png");
SetWindowIcon(icon);
UnloadImage(icon);

//Аудио
InitAudioDevice();
Sounds snd = {
    .music = LoadMusicStream("m.mp3"),
    .shoot = LoadSound("Shoot.mp3"),
    .hitEnemy = LoadSound("Hit.mp3"),
    .hitPlayer = LoadSound("HP.mp3"),
    .gameOver = LoadSound("GameOver.mp3"),
};
PlayMusicStream(snd.music);

//Фон
int    bgQueue[BG_SEGMENTS];
float bgOffset = 0.0f;
Background_Init(bgQueue);

//Таблицарекордов
Scoreboard scoreboard;
Scoreboard_Load(&scoreboard);

//Игровыеобъекты
Player    player;
Bullet    bullets[MAX_BULLETS] = { 0 };
Bullet    eneBullets[MAX_ENE_BULLETS] = { 0 };
Enemy     enemies[MAX_ENEMIES] = { 0 };
GameState gs = { 0 };
float     levelTimer = 0.0f;

//Начальныйэкран
GameScreen screen = SCREEN_MENU;

```

```

//ГЛАВНЫЙЦИКЛ
while (!WindowShouldClose()) {
    float dt = GetFrameTime();
UpdateMusicStream(snd.music);

    // Фон скроллится всегда (и в меню тоже)
Background_Update(bgQueue, &bgOffset, dt);

    // — Логикапозэкрану
    if (screen == SCREEN_MENU) {

        int choice = Menu_Update();
        if (choice == MENU_PLAY) {
            ResetGame(&player, bullets, eneBullets,
                enemies, &gs, &snd, &levelTimer);
            Background_Init(bgQueue);
            bgOffset = 0.0f;
            screen = SCREEN_GAME;
        }
        else if (choice == MENU_QUIT) {
break;

        }
        // MENU_SCORES — остаёмся в меню, там показывается мини-
таблица

    }

    else if (screen == SCREEN_GAME) {

        levelTimer = Level_Update(enemies, dt);
        Player_Update(&player, &bgOffset, dt);
        Bullets_Shoot(bullets, &player, &snd);
        Bullets_UpdatePlayer(bullets, dt);
        Bullets_UpdateEnemy(eneBullets, dt);
        Enemies_Update(enemies, eneBullets, player.x, player.y,
dt);

        gs.invincibleTimer -= dt;
        Collision_BulletsVsEnemies(bullets, enemies, &gs, &snd);
        Collision_EnemyBulletsVsPlayer(eneBullets, &player, &gs,
&snd);

```

```

Collision_EnemiesVsPlayer(enemies, &player, &gs, &snd);

if (gs.gameOver || gs.victory) {
    StopMusicStream(snd.music);
    screen = SCREEN_END;
}

}

else if (screen == SCREEN_END) {
// Ввод имени и сохранение обрабатываются внутри HUD_DrawEndScreen
}

//Отрисовка
BeginDrawing();
ClearBackground(BLACK);

if (screen == SCREEN_MENU) {
    Menu_Draw(bgQueue, bgOffset, tex.bg, tex.plane,
&scoreboard);
}

else {
    // Игра и экран конца – рисуем игровую сцену
Background_Draw(bgQueue, bgOffset, tex.bg);
    Bullets_DrawPlayer(bullets, tex.bullet);
    Bullets_DrawEnemy(eneBullets, tex.bullet);
    Enemies_Draw(enemies, tex.enemyA, tex.enemyB);
    Player_Draw(&player, tex.plane);
    HUD_Draw(&gs, tex.heart, levelTimer);

if (screen == SCREEN_END) {
    bool saved = HUD_DrawEndScreen(&gs, &scoreboard);

    if (saved) {
        if (IsKeyPressed(KEY_R)) {
            // Рестартсновавигру
            ResetGame(&player, bullets, eneBullets,
                enemies, &gs, &snd, &levelTimer);
            Background_Init(bgQueue);
            bgOffset = 0.0f;
            screen = SCREEN_GAME;
        }
    }
}
}

```

```

        }
        if (IsKeyPressed(KEY_ESCAPE)) {
            // Выход в меню
            HUD_ResetNameEntry();
            screen = SCREEN_MENU;
            PlayMusicStream(snd.music);
        }
    }
}

    EndDrawing();
}

//Выгрузка
if (snd.shoot.frameCount > 0) UnloadSound(snd.shoot);
if (snd.hitEnemy.frameCount > 0) UnloadSound(snd.hitEnemy);
if (snd.hitPlayer.frameCount > 0) UnloadSound(snd.hitPlayer);
if (snd.gameOver.frameCount > 0) UnloadSound(snd.gameOver);
UnloadMusicStream(snd.music);
CloseAudioDevice();

for (int i = 0; i < 3; i++) UnloadTexture(tex.bg[i]);
UnloadTexture(tex.plane);
UnloadTexture(tex.enemyA);
UnloadTexture(tex.enemyB);
UnloadTexture(tex.bullet);
UnloadTexture(tex.heart);

CloseWindow();
return 0;
}

```

## Файл anim.c

```
#include "game.h"

void Anim_Step(Animation* a) {
    float dt = GetFrameTime();
    a->duration -= dt;
    if (a->duration <= 0.0f) {
        a->duration = a->speed;
        a->cur++;
    }
    if (a->cur > a->last) a->cur = a->first;
}

Rectangle Anim_Frame(Animation* a, int numFrames) {
    int x = 0;
    int idx = a->cur % numFrames;
    __asm {
        mov  eax, idx
        imul eax, 59      ; x = idx * 59
        mov  x,  eax
    }
    return (Rectangle) { (float)x, 0, 59.0f, 64.0f };
}
```

## Файл background.c

```
#include "game.h"

void Background_Init(int queue[BG_SEGMENTS]) {
    for (int i = 0; i < BG_SEGMENTS; i++)
        queue[i] = rand() % 3;
}

void Background_Update(int queue[BG_SEGMENTS], float* offset, float
dt) {
    *offset += 150.0f * dt;
    if (*offset >= SCREEN_H) {
        *offset -= SCREEN_H;
        for (int i = 0; i < BG_SEGMENTS - 1; i++)
            queue[i] = queue[i + 1];
        queue[BG_SEGMENTS - 1] = rand() % 3;
    }
}

void Background_Draw(int queue[BG_SEGMENTS], float offset, Texture2D
textures[3]) {
    Rectangle src = { 0, 0,
        (float)textures[0].width,
        (float)textures[0].height };

    for (int i = 0; i < BG_SEGMENTS; i++) {
        Texture2D cur = textures[queue[i]];
        Rectangle dst = { 0, offset - SCREEN_H * i, SCREEN_W,
SCREEN_H };
        DrawTexturePro(cur, src, dst, (Vector2) { 0, 0 }, 0, WHITE);
    }
}
```

## Файл player.c

```
#include "game.h"

#define PLAYER_SPEED      300.0f
#define PLAYER_MAX_ROT    25.0f
#define PLAYER_ROT_SPEED  50.0f
#define PLAYER_SHADOW_SPEED 0.2f

void Player_Init(Player* p) {
    p->x = SCREEN_W / 2.0f;
    p->y = SCREEN_H - 200.0f;
    p->shadowX = p->x;
    p->shadowY = p->y;
    p->speed = PLAYER_SPEED;
    p->rotation = 0.0f;
    p->shootCooldown = 0.0f;
    p->anim = (Animation){ 0, 3, 0, 0.05f, 0.1f };
}

void Player_Update(Player* p, float* bgOffset, float dt) {

    if (IsKeyDown(KEY_W) || IsKeyDown(KEY_UP)) { p->y -= p->speed *
dt; *bgOffset += 1.0f; }
    if (IsKeyDown(KEY_S) || IsKeyDown(KEY_DOWN)) { p->y += p->speed *
dt; *bgOffset -= 1.0f; }

    if (IsKeyDown(KEY_A) || IsKeyDown(KEY_LEFT)) {
        p->x -= p->speed * dt;
        p->rotation -= PLAYER_ROT_SPEED * dt * (p->rotation > 0 ?
2.0f : 1.0f);
    }
    else if (IsKeyDown(KEY_D) || IsKeyDown(KEY_RIGHT)) {
        p->x += p->speed * dt;
        p->rotation += PLAYER_ROT_SPEED * dt * (p->rotation < 0 ?
2.0f : 1.0f);
    }
    else {
        if (p->rotation > 0) { p->rotation -= 2 * PLAYER_ROT_SPEED *
dt; if (p->rotation < 0) p->rotation = 0; }
```

```

        else if (p->rotation < 0) { p->rotation += 2 *
PLAYER_ROT_SPEED * dt; if (p->rotation > 0) p->rotation = 0; }
    }
    if (p->rotation > PLAYER_MAX_ROT) p->rotation = PLAYER_MAX_ROT;
    if (p->rotation < -PLAYER_MAX_ROT) p->rotation = -PLAYER_MAX_ROT;
    if (p->x < 26) p->x = 26;
    if (p->x > SCREEN_W - 26) p->x = SCREEN_W - 26;
    if (p->y < 256) p->y = 256;
    if (p->y > SCREEN_H - 96) p->y = SCREEN_H - 96;
    p->shadowX += (p->x - p->shadowX) * PLAYER_SHADOW_SPEED;
    p->shadowY += (p->y - p->shadowY) * PLAYER_SHADOW_SPEED;
    p->shootCooldown -= dt;
    Anim_Step(&p->anim);
}

void Player_Draw(Player* p, Texture2D tex) {
    Rectangle frame = Anim_Frame(&p->anim, 4);

    int shadowLayers = 4;
    float shadowStep = 2.0f;
    int shadowAlpha = 25;
    for (int i = shadowLayers; i >= 1; i--) {
        Rectangle sd = {
            p->shadowX + shadowStep * i,
            p->shadowY + shadowStep * i,
            100, 100
        };
        DrawShadow(tex, frame, sd, (Vector2) { 50, 50 },
            p->rotation, shadowLayers, shadowStep, shadowAlpha);
    }

    Rectangle dst = { p->x, p->y, 96, 96 };
    DrawTexturePro(tex, frame, dst, (Vector2) { 50, 50 }, p-
>rotation, WHITE);
}

```

## Файл bullets.c

```
#include "game.h"

#define BULLET_SPEED      700.0f
#define SHOOT_INTERVAL    0.15f

// Выстрелил игрок
void Bullets_Shoot(Bullet bullets[MAX_BULLETS], Player* p, Sounds*
snd) {
    if (!IsKeyDown(KEY_SPACE) || p->shootCooldown > 0.0f) return;

    p->shootCooldown = SHOOT_INTERVAL;
    if (snd->shoot.frameCount > 0) PlaySound(snd->shoot);

    float offsets[] = { -47.0f, 26.0f }; // левый и правый ствол
    for (int side = 0; side < 2; side++) {
        for (int i = 0; i < MAX_BULLETS; i++) {
            if (!bullets[i].active) {
                bullets[i].active = true;
                bullets[i].x = p->x + offsets[side];
                bullets[i].y = p->y + 10.0f;
                bullets[i].speedX = 0.0f;
                bullets[i].speedY = -BULLET_SPEED;
                break;
            }
        }
    }
}

// Обновление пуль игрока
void Bullets_UpdatePlayer(Bullet bullets[MAX_BULLETS], float dt) {
    for (int i = 0; i < MAX_BULLETS; i++) {
        if (!bullets[i].active) continue;
        bullets[i].y += bullets[i].speedY * dt;
        if (bullets[i].y < -32) bullets[i].active = false;
    }
}
```

```

// Обновление пуль врагов
void Bullets_UpdateEnemy(Bullet bullets[MAX_ENE_BULLETS], float dt) {
    for (int i = 0; i < MAX_ENE_BULLETS; i++) {
        if (!bullets[i].active) continue;
        bullets[i].x += bullets[i].speedX * dt;
        bullets[i].y += bullets[i].speedY * dt;
        if (bullets[i].y > SCREEN_H + 32 || bullets[i].y < -32 ||
            bullets[i].x > SCREEN_W + 32 || bullets[i].x < -32)
            bullets[i].active = false;
    }
}

// Отрисовка пуль игрока
void Bullets_DrawPlayer(Bullet bullets[MAX_BULLETS], Texture2D tex) {
    Rectangle src = { 0, 0, (float)tex.width, (float)tex.height };
    for (int i = 0; i < MAX_BULLETS; i++) {
        if (!bullets[i].active) continue;
        Rectangle dst = { bullets[i].x, bullets[i].y, 20, 36 };
        DrawShadow(tex, src, dst, (Vector2) { 0, 10 }, 0, 3, 1.5f,
40);

        DrawTexturePro(tex, src, dst, (Vector2) { 0, 10 }, 0, WHITE);
    }
}

// Отрисовка пуль врагов
void Bullets_DrawEnemy(Bullet bullets[MAX_ENE_BULLETS], Texture2D
tex) {
    Rectangle src = { 0, 0, (float)tex.width, (float)tex.height };
    for (int i = 0; i < MAX_ENE_BULLETS; i++) {
        if (!bullets[i].active) continue;
        float angle = atan2f(bullets[i].speedX, -bullets[i].speedY) *
RAD2DEG;

        Rectangle dst = { bullets[i].x, bullets[i].y, 20, 36 };
        DrawShadow(tex, src, dst, (Vector2) { 10, 18 }, angle, 3,
1.5f, 40);

        DrawTexturePro(tex, src, dst, (Vector2) { 10, 18 }, angle,
(Color) {
            255, 80, 80, 255
        });
    }
}

```



## Файл enemies.c

```
#include "game.h"

// Спавн
void Enemy_Spawn(Enemy enemies[MAX_ENEMIES], EnemyType type, float x)
{
    for (int i = 0; i < MAX_ENEMIES; i++) {
        if (enemies[i].active) continue;

        Enemy* e = &enemies[i];
        e->active = true;
        e->type = type;
        e->x = x;
        e->baseX = x;
        e->waveTime = 0.0f;
        e->shadowX = x;
        e->shadowY = -64.0f;
        e->reachedTarget = false;
        e->deathTimer = 0.0f;

        if (x < 0) e->x = 0;
        if (x > SCREEN_W - 64) e->x = SCREEN_W - 64;

        switch (type) {
            case ENEMY_KAMIKAZE:
                e->y = -64.0f;
                e->hp = 2;
                e->maxHp = 2;
                e->speed = 450.0f + (float)(rand() % 100);
                e->shootTimer = 9999.0f;
                e->shootInterval = 9999.0f;
                e->anim = (Animation){ 0, 3, 0, 0.08f, 0.08f };
                break;

            case ENEMY_FIGHTER:
                e->y = -64.0f;
                e->hp = 4;
                e->maxHp = 4;
                e->speed = 200.0f;
```

```

        e->targetY = 80.0f + (float)(rand() % 200);
        e->shootTimer = 1.0f;
        e->shootInterval = 2.0f;
        e->anim = (Animation){ 0, 3, 0, 0.10f, 0.10f };
        break;

    case ENEMY_BOSS:
        e->y = -140.0f;
        e->hp = 80;
        e->maxHp = 80;
        e->speed = 120.0f;
        e->targetY = 60.0f;
        e->shootTimer = 0.8f;
        e->shootInterval = 0.8f;
        e->bossDir = 1.0f;
        e->bossTimer = 2.0f;
        e->anim = (Animation){ 0, 3, 0, 0.06f, 0.06f };
        break;
    }
    break;
}

// Обновление всех врагов
void Enemies_Update(Enemy enemies[MAX_ENEMIES],
    Bullet eneBullets[MAX_ENE_BULLETS],
    float planeX, float planeY, float dt)
{
    for (int i = 0; i < MAX_ENEMIES; i++) {
        Enemy* e = &enemies[i];

        // Анимация смерти
        if (e->deathTimer > 0.0f) {
            e->deathTimer -= dt;
            if (e->deathTimer <= 0.0f)
                e->active = false;
            continue;
        }
        if (!e->active) continue;

        Anim_Step(&e->anim);
    }
}

```

```

e->waveTime += dt;

// Камикадзе
if (e->type == ENEMY_KAMIKAZE) {
    e->y += e->speed * dt;
    e->x = e->baseX + sinf(e->waveTime * 3.0f) * 20.0f;
    if (e->y > SCREEN_H + 64) e->active = false;
}
// Файтер
else if (e->type == ENEMY_FIGHTER) {
    if (!e->reachedTarget) {
        e->y += e->speed * dt;
        if (e->y >= e->targetY) {
            e->y = e->targetY;
            e->reachedTarget = true;
            e->baseX = e->x;
            e->waveTime = 0.0f;
        }
    }
    else {
        e->x = e->baseX + sinf(e->waveTime * 1.5f) * 120.0f;
        e->y += 20.0f * dt;
    }
    if (e->y > SCREEN_H + 64) { e->active = false; continue;
}

// Стрельба строго вниз
e->shootTimer -= dt;
if (e->shootTimer <= 0.0f && e->reachedTarget) {
    e->shootTimer = e->shootInterval;
    for (int b = 0; b < MAX_ENE_BULLETS; b++) {
        if (eneBullets[b].active) continue;
        eneBullets[b].active = true;
        eneBullets[b].x = e->x + 32;
        eneBullets[b].y = e->y + 32;
        eneBullets[b].speedX = 0.0f;
        eneBullets[b].speedY = 500.0f;
        break;
    }
}
}
// Босс

```

```

else if (e->type == ENEMY_BOSS) {
    if (!e->reachedTarget) {
        e->y += e->speed * dt;
        if (e->y >= e->targetY) {
            e->y = e->targetY;
            e->reachedTarget = true;
        }
    }
    else {
        e->bossTimer -= dt;
        if (e->bossTimer <= 0.0f) {
            e->bossDir *= -1.0f;
            e->bossTimer = 1.5f + (float)(rand() % 20) /
10.0f;

        }
        e->x += e->bossDir * 160.0f * dt;
        if (e->x < 0) { e->x = 0;                e->bossDir =
1.0f; }

        if (e->x > SCREEN_W - 192) { e->x = SCREEN_W - 192;
e->bossDir = -1.0f; }
    }

    // Трёхпулевойвееер
    e->shootTimer -= dt;
    if (e->shootTimer <= 0.0f && e->reachedTarget) {
        e->shootTimer = e->shootInterval;
        float angles[] = { -15.0f, 0.0f, 15.0f };
        for (int a = 0; a < 3; a++) {
            float ang = angles[a] * DEG2RAD;
            float bs = 240.0f;
            float nx = sinf(ang);
            float ny = cosf(ang);
            for (int b = 0; b < MAX_ENE_BULLETS; b++) {
                if (eneBullets[b].active) continue;
                eneBullets[b].active = true;
                eneBullets[b].x = e->x + 96;
                eneBullets[b].y = e->y + 96;
                eneBullets[b].speedX = nx * bs;
                eneBullets[b].speedY = ny * bs;
                break;
            }
        }
    }
}

```

```

    }
}
}

// Отрисовка всех врагов
void Enemies_Draw(Enemy enemies[MAX_ENEMIES],
    Texture2D texA, Texture2D texB)
{
    for (int i = 0; i < MAX_ENEMIES; i++) {
        Enemy* e = &enemies[i];
        if (!e->active && e->deathTimer <= 0.0f) continue;

        // Мигание при смерти
        if (e->deathTimer > 0.0f && (int)(e->deathTimer * 10) % 2 ==
0)
            continue;

        Rectangle frame = Anim_Frame(&e->anim, 4);

        if (e->type == ENEMY_BOSS) {
            Rectangle dst = { e->x, e->y, 192, 192 };
            DrawShadow(texA, frame, dst, (Vector2) { 0, 0 }, 0, 4,
2.5f, 60);

            DrawTexturePro(texA, frame, dst, (Vector2) { 0, 0 }, 0,
                (Color) {
                    255, 160, 160, 255
                });

            // HP-бар босса
            float hpFrac = e->hp / e->maxHp;
            DrawRectangle(SCREEN_W / 2 - 200, 20, 400, 14, DARKGRAY);
            DrawRectangle(SCREEN_W / 2 - 200, 20, (int)(400 *
hpFrac), 14,
                hpFrac > 0.5f ? GREEN : hpFrac > 0.25f ? YELLOW :
RED);

            DrawText("BOSS", SCREEN_W / 2 - 22, 18, 14, WHITE);
        }
        else {
            Texture2D tex = (e->type == ENEMY_KAMIKAZE) ? texB :
texA;

```

```

        Color tint = (e->type == ENEMY_FIGHTER)
            ? (Color) { 180, 220, 255, 255 } : WHITE;
        Rectangle dst = { e->x, e->y, 64, 64 };
        DrawShadow(tex, frame, dst, (Vector2) { 0, 0 }, 0, 3,
1.5f, 40);

        DrawTexturePro(tex, frame, dst, (Vector2) { 0, 0 }, 0,
tint);

        // HP-полоска
        float hpFrac = e->hp / e->maxHp;
        DrawRectangle((int)e->x, (int)e->y - 8, 64, 5, DARKGRAY);
        DrawRectangle((int)e->x, (int)e->y - 8, (int)(64 *
hpFrac), 5, GREEN);
    }
}
}

```

## Файл level.c

```
#include "game.h"

LevelEvent gLevel[] = {

    // Волна 1 – знакомство с камикадзе (0-30 сек)
    { 3.0f, 200, ENEMY_KAMIKAZE },
        { 4.0f, 600, ENEMY_KAMIKAZE },
        { 5.5f, 400, ENEMY_KAMIKAZE },
        { 7.0f, 150, ENEMY_KAMIKAZE },
        { 7.5f, 600, ENEMY_KAMIKAZE },
        { 9.0f, 350, ENEMY_KAMIKAZE },
        { 10.0f, 550, ENEMY_KAMIKAZE },

    // Волна 2 – первыефайтеры (30-60 сек)
    { 15.0f, 300, ENEMY_FIGHTER },
    { 15.5f, 600, ENEMY_FIGHTER },
    { 17.0f, 150, ENEMY_KAMIKAZE },
    { 17.5f, 650, ENEMY_KAMIKAZE },
    { 19.0f, 400, ENEMY_FIGHTER },
    { 21.0f, 250, ENEMY_KAMIKAZE },
    { 21.5f, 500, ENEMY_KAMIKAZE },

    // Волна 3 – смешанная (60-100 сек)
    { 30.0f, 200, ENEMY_FIGHTER },
    { 30.5f, 700, ENEMY_FIGHTER },
    { 31.0f, 450, ENEMY_KAMIKAZE },
    { 33.0f, 100, ENEMY_KAMIKAZE },
    { 33.5f, 650, ENEMY_KAMIKAZE },
    { 35.0f, 400, ENEMY_FIGHTER },
    { 36.0f, 300, ENEMY_KAMIKAZE },
    { 37.0f, 600, ENEMY_KAMIKAZE },
    { 38.0f, 500, ENEMY_FIGHTER },

    // Волна 4 – плотный натиск (100-160 сек)
    { 45.0f, 150, ENEMY_KAMIKAZE },
        { 45.3f, 400, ENEMY_KAMIKAZE },
        { 45.6f, 650, ENEMY_KAMIKAZE },
        { 47.0f, 300, ENEMY_FIGHTER },
        { 47.5f, 550, ENEMY_FIGHTER },
        { 49.0f, 200, ENEMY_KAMIKAZE },
```

```

    { 50.0f, 700, ENEMY_KAMIKAZE },
    { 51.0f, 450, ENEMY_FIGHTER },
    { 53.0f, 100, ENEMY_KAMIKAZE },
    { 53.5f, 400, ENEMY_KAMIKAZE },
    { 54.0f, 650, ENEMY_KAMIKAZE },
    { 56.0f, 350, ENEMY_FIGHTER },
    { 57.0f, 600, ENEMY_FIGHTER },

    // Волна 5 – финальный штурм перед боссом (160-240 сек)
    { 65.0f, 200, ENEMY_FIGHTER },
    { 65.5f, 400, ENEMY_FIGHTER },
    { 66.0f, 650, ENEMY_FIGHTER },
    { 68.0f, 100, ENEMY_KAMIKAZE },
    { 68.2f, 300, ENEMY_KAMIKAZE },
    { 68.4f, 500, ENEMY_KAMIKAZE },
    { 68.6f, 600, ENEMY_KAMIKAZE },
    { 70.0f, 250, ENEMY_FIGHTER },
    { 71.0f, 550, ENEMY_FIGHTER },
    { 72.0f, 400, ENEMY_KAMIKAZE },
    { 74.0f, 200, ENEMY_KAMIKAZE },
    { 74.5f, 650, ENEMY_KAMIKAZE },
    { 76.0f, 350, ENEMY_FIGHTER },
    { 78.0f, 500, ENEMY_KAMIKAZE },
    { 79.0f, 150, ENEMY_FIGHTER },
    { 80.0f, 650, ENEMY_FIGHTER },

    // Брейк перед боссом
    { 88.0f, 300, ENEMY_KAMIKAZE },
    { 88.5f, 500, ENEMY_KAMIKAZE },
    { 90.0f, 200, ENEMY_FIGHTER },
    { 93.5f, 400, ENEMY_FIGHTER },
    { 96.0f, 650, ENEMY_FIGHTER },
    { 100.0f, 100, ENEMY_KAMIKAZE },
    { 103.2f, 300, ENEMY_KAMIKAZE },
    { 106.4f, 500, ENEMY_KAMIKAZE },
    { 109.6f, 600, ENEMY_KAMIKAZE },
    { 111.0f, 250, ENEMY_FIGHTER },
    { 120.0f, 550, ENEMY_FIGHTER },
    { 123.0f, 400, ENEMY_KAMIKAZE },
    { 126.0f, 200, ENEMY_KAMIKAZE },
    { 128.5f, 750, ENEMY_KAMIKAZE },
    { 130.0f, 350, ENEMY_FIGHTER },

```

```

{133.0f, 500, ENEMY_KAMIKAZE },
{136.0f, 150, ENEMY_FIGHTER },
{139.0f, 650, ENEMY_FIGHTER },
{142.0f, 200, ENEMY_FIGHTER },
{144.0f, 200, ENEMY_KAMIKAZE },
{146.0f, 200, ENEMY_FIGHTER },
{148.0f, 200, ENEMY_KAMIKAZE },
{154.0f, 200, ENEMY_FIGHTER },
{156.0f, 200, ENEMY_FIGHTER },
{160.0f, 200, ENEMY_KAMIKAZE },
{162.0f, 650, ENEMY_FIGHTER },
{164.0f, 200, ENEMY_KAMIKAZE },
{166.0f, 200, ENEMY_KAMIKAZE },
{168.0f, 200, ENEMY_KAMIKAZE },

```

```

// Большая волна камикадзе

```

```

{170.0f, 200, ENEMY_KAMIKAZE },
{172.0f, 250, ENEMY_FIGHTER },
{174.0f, 250, ENEMY_KAMIKAZE },
{175.0f, 300, ENEMY_KAMIKAZE },
{178.0f, 200, ENEMY_FIGHTER },
{180.0f, 500, ENEMY_KAMIKAZE },
{182.0f, 200, ENEMY_KAMIKAZE },
{184.0f, 400, ENEMY_FIGHTER },
{186.0f, 200, ENEMY_KAMIKAZE },
{188.0f, 100, ENEMY_KAMIKAZE },
{190.0f, 400, ENEMY_FIGHTER },
{195.0f, 200, ENEMY_KAMIKAZE },
{200.0f, 700, ENEMY_KAMIKAZE },
{205.0f, 400, ENEMY_FIGHTER },
{208.0f, 200, ENEMY_KAMIKAZE },
{211.0f, 700, ENEMY_KAMIKAZE },
{214.0f, 400, ENEMY_FIGHTER },
{216.0f, 200, ENEMY_KAMIKAZE },
{220.0f, 600, ENEMY_KAMIKAZE },
{222.0f, 400, ENEMY_FIGHTER },
{224.0f, 200, ENEMY_KAMIKAZE },
{226.0f, 700, ENEMY_KAMIKAZE },
{230.0f, 400, ENEMY_FIGHTER },
{232.0f, 200, ENEMY_KAMIKAZE },
{235.0f, 600, ENEMY_KAMIKAZE },
{240.0f, 400, ENEMY_FIGHTER },

```

```

        {242.0f, 200, ENEMY_KAMIKAZE },
        {245.0f, 600, ENEMY_KAMIKAZE },
        {250.0f, 400, ENEMY_FIGHTER  },
        {252.0f, 200, ENEMY_KAMIKAZE },
        {255.0f, 600, ENEMY_KAMIKAZE },

        // БОСС (~295 сек)
        {295.0f, 384, ENEMY_BOSS      },
};

int gTotalEvents = sizeof(gLevel) / sizeof(gLevel[0]);

// Состояние прогресса уровня
static int   sNextEvent = 0;
static float sLevelTimer = 0.0f;

void Level_Init(void) {
    sNextEvent = 0;
    sLevelTimer = 0.0f;
}

float Level_Update(Enemy enemies[MAX_ENEMIES], float dt) {
    sLevelTimer += dt;

    while (sNextEvent < gTotalEvents && sLevelTimer >=
gLevel[sNextEvent].time)
    {
        Enemy_Spawn(enemies,
            gLevel[sNextEvent].type,
            gLevel[sNextEvent].x);
        sNextEvent++;
    }

    return sLevelTimer;
}

```

## Файл collision.c

```
#include "game.h"

// Пули игрока враги
void Collision_BulletsVsEnemies(Bullet bullets[MAX_BULLETS],
    Enemy enemies[MAX_ENEMIES],
    GameState* gs, Sounds* snd)
{
    for (int b = 0; b < MAX_BULLETS; b++) {
        if (!bullets[b].active) continue;
        for (int e = 0; e < MAX_ENEMIES; e++) {
            if (!enemies[e].active) continue;
            if (enemies[e].deathTimer > 0.0f) continue;

            float ew = (enemies[e].type == ENEMY_BOSS) ? 192.0f :
64.0f;

            float eh = (enemies[e].type == ENEMY_BOSS) ? 192.0f :
64.0f;

            bool hit = bullets[b].x < enemies[e].x + ew &&
                bullets[b].x + 16 > enemies[e].x &&
                bullets[b].y < enemies[e].y + eh &&
                bullets[b].y + 32 > enemies[e].y;

            if (!hit) continue;

            bullets[b].active = false;
            enemies[e].hp -= 1.0f;
            if (snd->hitEnemy.frameCount > 0) PlaySound(snd-
>hitEnemy);

            if (enemies[e].hp <= 0 && enemies[e].deathTimer <= 0.0f)
            {
                enemies[e].deathTimer = 0.5f;
                if (enemies[e].type == ENEMY_BOSS) {
                    gs->score += 5000;
                    gs->bossAlive = false;
                    gs->victory = true;
                }
            }
        }
    }
}
```

```

        }
        else {
            gs->score += (enemies[e].type == ENEMY_KAMIKAZE)
? 100 : 200;
        }
    }
}

// Пули врагов и игрок
void Collision_EnemyBulletsVsPlayer(Bullet
eneBullets[MAX_ENE_BULLETS],
Player* p, GameState* gs, Sounds* snd)
{
    if (gs->gameOver || gs->invincibleTimer > 0.0f) return;

    Rectangle hitbox = { p->x - 20, p->y - 20, 56, 56 };

    for (int b = 0; b < MAX_ENE_BULLETS; b++) {
        if (!eneBullets[b].active) continue;

        bool hit = eneBullets[b].x > hitbox.x &&
            eneBullets[b].x < hitbox.x + hitbox.width &&
            eneBullets[b].y > hitbox.y &&
            eneBullets[b].y < hitbox.y + hitbox.height;

        if (!hit) continue;

        eneBullets[b].active = false;
        gs->playerHp--;
        gs->invincibleTimer = 1.5f;
        if (snd->hitPlayer.frameCount > 0) PlaySound(snd->hitPlayer);

        if (gs->playerHp <= 0) {
            gs->gameOver = true;
            if (snd->gameOver.frameCount > 0) PlaySound(snd-
>gameOver);
        }

        return; // один хит за кадр
    }
}

```

```

    }
}

// Враг и игрок (таран)
void Collision_EnemiesVsPlayer(Enemy enemies[MAX_ENEMIES],
    Player* p, GameState* gs, Sounds* snd)
{
    if (gs->gameOver || gs->invincibleTimer > 0.0f) return;

    Rectangle playerHitbox = { p->x - 20, p->y - 20, 56, 56 };

    for (int e = 0; e < MAX_ENEMIES; e++) {
        if (!enemies[e].active) continue;
        if (enemies[e].deathTimer > 0) continue;

        float ew = (enemies[e].type == ENEMY_BOSS) ? 192.0f : 64.0f;
        float eh = (enemies[e].type == ENEMY_BOSS) ? 192.0f : 64.0f;
        Rectangle enemyHitbox = { enemies[e].x, enemies[e].y, ew, eh
};

        if (!CheckCollisionRecs(playerHitbox, enemyHitbox)) continue;

        gs->playerHp--;
        gs->invincibleTimer = 1.5f;
        if (snd->hitPlayer.frameCount > 0) PlaySound(snd->hitPlayer);

        if (enemies[e].type == ENEMY_BOSS) {
            enemies[e].hp -= 10.0f;
            if (enemies[e].hp <= 0 && enemies[e].deathTimer <= 0.0f)
            {
                enemies[e].deathTimer = 0.5f;
                gs->score += 5000;
                gs->bossAlive = false;
                gs->victory = true;
            }
        }
        else {
            enemies[e].hp = 0;
            enemies[e].deathTimer = 0.5f;

```

```

        gs->score += (enemies[e].type == ENEMY_KAMIKAZE) ? 100 :
200;
    }

    if (gs->playerHp <= 0) {
        gs->gameOver = true;
        if (snd->gameOver.frameCount > 0) PlaySound(snd-
>gameOver);
    }

    return; // один хит за кадр
}
}

```

## Файл hud.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "game.h"
#include <string.h>

//Внутреннее состояние ввода имени
static char sName[PLAYER_NAME_LEN] = { 0 };
static int sNameLen = 0;
static bool sSubmitted = false;
static int sRankIdx = -1; // местов таблице (-1 =
ещё неизвестно)

// Сброс при новой игре
void HUD_ResetNameEntry(void) {
    memset(sName, 0, sizeof(sName));
    sNameLen = 0;
    sSubmitted = false;
    sRankIdx = -1;
}

// Основной HUD (во время игры)
void HUD_Draw(GameState* gs, Texture2D heartTex, float lt) {
    // Очки
    DrawText(TextFormat("Score: %d", gs->score), 10, 10, 28, WHITE);

    // Сердечки HP
    Rectangle heartSrc = { 0, 0,
        (float)heartTex.width,
        (float)heartTex.height };
    for (int h = 0; h < 5; h++) {
        Color hc = (h < gs->playerHp) ? RED : DARKGRAY;
        DrawTexturePro(heartTex, heartSrc,
            (Rectangle) {
                8 + h * 25, 40, 24, 24
            },
            (Vector2) {
                0, 0
            }, 0, hc);
    }
}
```

```

        // Таймер
        int timeLeft = 300 - (int)lt;
        if (timeLeft < 0) timeLeft = 0;
        DrawText(TextFormat("Time: %d:%02d", timeLeft / 60, timeLeft %
60),
                SCREEN_W - 160, 10, 28,
                (timeLeft < 30) ? RED : WHITE);

        // Миганиеэкранаприпопадании
        if (gs->invincibleTimer > 0.0f && (int)(gs->invincibleTimer * 8)
% 2 == 0)
                DrawRectangle(0, 0, SCREEN_W, SCREEN_H, (Color) { 255, 0, 0,
40 });
    }
    bool HUD_DrawEndScreen(GameState* gs, Scoreboard* sb) {
        // Затемнение
        Color overlay = gs->victory
                ? (Color) { 0, 0, 0, 160 }
                : (Color) { 0, 0, 0, 190 };
        DrawRectangle(0, 0, SCREEN_W, SCREEN_H, overlay);

        // Заголовок
        if (gs->victory) {
            DrawText("VICTORY!",
                    SCREEN_W / 2 - MeasureText("VICTORY!", 64) / 2,
                    120, 64, GOLD);
        }
        else {
            DrawText("GAME OVER",
                    SCREEN_W / 2 - MeasureText("GAME OVER", 64) / 2,
                    120, 64, RED);
        }

        DrawText(TextFormat("Score: %d", gs->score),
                SCREEN_W / 2 - MeasureText(TextFormat("Score: %d", gs-
>score), 30) / 2,
                200, 30, WHITE);

        if (!sSubmitted) {
            //Вводимени

```

```

        DrawText("Enter a name:",
            SCREEN_W / 2 - 120, 260, 26, LIGHTGRAY);

// Обработка клавиш
int key = GetCharPressed();
while (key > 0) {
    if (key >= 32 && key <= 126 && sNameLen < PLAYER_NAME_LEN
- 1) {

        sName[sNameLen++] = (char)key;
        sName[sNameLen] = '\0';
    }
    key = GetCharPressed();
}
if (IsKeyPressed(KEY_BACKSPACE) && sNameLen > 0)
    sName[--sNameLen] = '\0';

// Поле ввода
const char* displayName = (sNameLen == 0) ? "____" : sName;
int nameW = MeasureText(displayName, 32);
int boxX = SCREEN_W / 2 - 160;
int boxY = 295;
DrawRectangle(boxX, boxY, 320, 44, (Color) { 20, 20, 60, 220
});

DrawRectangleLines(boxX, boxY, 320, 44, (Color) { 100, 180,
255, 200 });

DrawText(displayName, SCREEN_W / 2 - nameW / 2, boxY + 8, 32,
    sNameLen == 0 ? GRAY : YELLOW);

// Мигающий курсор
if ((int)(GetTime() * 2) % 2 == 0 && sNameLen <
PLAYER_NAME_LEN - 1)
    DrawText("|", SCREEN_W / 2 + nameW / 2 + 2, boxY + 8, 32,
YELLOW);

// Подсказка
if (sNameLen > 0) {
    DrawText("[ENTER] - save",
        SCREEN_W / 2 - MeasureText("[ENTER] - save", 20) / 2,
        350, 20, DARKGREEN);
}
else {

```

```

        DrawText("(enter at least 1 character)",
            SCREEN_W / 2 - MeasureText("(enter at least 1
character)", 18) / 2,
            350, 18, DARKGRAY);
    }

    if (IsKeyPressed(KEY_ENTER) && sNameLen > 0) {
        sRankIdx = Scoreboard_Submit(sb, sName, gs->score, gs-
>victory);
        sSubmitted = true;
    }

    // Таблица (только просмотр, без подсветки)
    Scoreboard_Draw(sb, -1);
    return false;

}

else {
    //Таблица с подсветкой новой записи
    const char* msg = (sRankIdx >= 0)
        ? TextFormat("Your place: #%d - [R] restart", sRankIdx
+ 1)
        : "Not included in TOP-10 - [R] restart";
    DrawText(msg,
        SCREEN_W / 2 - MeasureText(msg, 22) / 2,
        260, 22,
        sRankIdx >= 0 ? GOLD : GRAY);

    Scoreboard_Draw(sb, sRankIdx);
    return true;
}
}

```

## Файл menu.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "game.h"
#include <math.h>

//Внутреннее состояние
static int    sCursor = 0;           // выбранный пункт
static float  sBlinkTime = 0.0f;     // для мигания курсора
static float  sDemoPlaneX = 200.0f;  // декоративный самолёт
static float  sDemoPlaneY = 400.0f;

static const char* sLabels[MENU_COUNT] = {
    "Play",
    "Records",
    "Exit",
};

//Обновление
int Menu_Update(void) {
    float dt = GetFrameTime();
    sBlinkTime += dt;

    // Декоративный самолёт летит по синусоиде
    sDemoPlaneX += 90.0f * dt;
    sDemoPlaneY = 420.0f + sinf(sDemoPlaneX * 0.008f) * 60.0f;
    if (sDemoPlaneX > SCREEN_W + 80) sDemoPlaneX = -80.0f;

    // Навигация
    if (IsKeyPressed(KEY_DOWN) || IsKeyPressed(KEY_S)) {
        sCursor = (sCursor + 1) % MENU_COUNT;
        sBlinkTime = 0.0f;
    }
    if (IsKeyPressed(KEY_UP) || IsKeyPressed(KEY_W)) {
        sCursor = (sCursor - 1 + MENU_COUNT) % MENU_COUNT;
        sBlinkTime = 0.0f;
    }

    // Подтверждение
```

```

        if (IsKeyPressed(KEY_ENTER) || IsKeyPressed(KEY_SPACE))
return sCursor;

        return -1;
}

//Вспомогательная: кнопка меню
static void DrawMenuButton(const char* label, int x, int y,
        bool selected, float blinkTime)
{
    int    fontSize = 36;
    int    w = MeasureText(label, fontSize);
    Color bg = selected
        ? (Color) { 20, 60, 120, 210 }
        : (Color) { 10, 20, 50, 160 };
    Color border = selected
        ? (Color) { 100, 200, 255, 255 }
        : (Color) { 40, 80, 140, 180 };
    Color textCol = selected ? YELLOW : (Color) { 180, 220, 255, 255
};

    int bw = 320, bh = 52;
    int bx = x - bw / 2;
    int by = y - bh / 2;

    DrawRectangle(bx, by, bw, bh, bg);
    DrawRectangleLines(bx, by, bw, bh, border);

    // Мигающий маркер слева
    if (selected && (int)(blinkTime * 3) % 2 == 0)
        DrawText(">", bx + 12, by + 8, fontSize, (Color) { 100, 200,
255, 255 });

    DrawText(label, x - w / 2, by + 8, fontSize, textCol);
}

//Отрисовка
void Menu_Draw(int bgQueue[BG_SEGMENTS], float bgOffset,
    Texture2D bgTex[3], Texture2D planeTex,
    const Scoreboard* sb)

```

```

{
    // Фон
    Background_Draw(bgQueue, bgOffset, bgTex);

    // Полупрозрачный оверлей
    DrawRectangle(0, 0, SCREEN_W, SCREEN_H, (Color) { 0, 0, 20, 160
});

    // — Заголовок —
    const char* title = "1942";
    int tw = MeasureText(title, 96);
    // Тень заголовка
    DrawText(title, SCREEN_W / 2 - tw / 2 + 4, 84, 96, (Color) { 0,
0, 0, 180 });
    DrawText(title, SCREEN_W / 2 - tw / 2, 80, 96, GOLD);

    // Подзаголовок
    const char* sub = "Battle of the planes";
    int sw = MeasureText(sub, 22);
    DrawText(sub, SCREEN_W / 2 - sw / 2, 185, 22, (Color) { 200, 220,
255, 200 });

    // Декоративная линия
    DrawRectangle(SCREEN_W / 2 - 200, 218, 400, 2, (Color) { 100,
180, 255, 120 });

    // — Пункты меню —
    int baseY = 340;
    int step = 76;
    for (int i = 0; i < MENU_COUNT; i++)
        DrawMenuButton(sLabels[i],
            SCREEN_W / 2, baseY + i * step,
            i == sCursor, sBlinkTime);

    // Подсказка управления
    DrawText("[W/S] - select      [ENTER] - confirm",
        SCREEN_W / 2 - MeasureText("[W/S] или [↑↓] - выбор      [ENTER]
- подтвердить", 16) / 3,
        SCREEN_H - 40, 16, (Color) { 120, 160, 200, 180 });

```

```

        //Мини-таблица рекордов (если выбран пункт РЕКОРДЫ)
if (sCursor == MENU_SCORES && sb->count > 0) {
    int px = SCREEN_W / 2 - 220;
    int py = 570;
    int pw = 440, ph = 30 + sb->count * 28 + 10;
    DrawRectangle(px, py, pw, ph, (Color) { 0, 0, 30, 220 });
    DrawRectangleLines(px, py, pw, ph, (Color) { 100, 180, 255,
160 });

    DrawText("TOP 10", px + pw / 2 - MeasureText("TOP 10", 18) /
2,
        py + 6, 18, GOLD);
    for (int i = 0; i < sb->count && i < 5; i++) {
        int ry = py + 32 + i * 28;
        Color nc = (sb->entries[i].victory) ? GREEN : RED;
        DrawText(TextFormat("%d.", i + 1), px + 10, ry, 20,
GRAY);

        DrawText(sb->entries[i].name, px + 40, ry, 20, WHITE);
        const char* sc = TextFormat("%d", sb->entries[i].score);
        DrawText(sc, px + pw - MeasureText(sc, 20) - 10, ry, 20,
nc);
    }
}
}

```

## Файл scoreboard.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "game.h"
#include <stdio.h>
#include <string.h>

//Загрузкаизфайла
void Scoreboard_Load(Scoreboard* sb) {
    sb->count = 0;

    FILE* f = fopen(SCOREBOARD_FILE, "rb");
    if (!f) return; // файла ещё нет — начнём с пустой таблицы

    fread(&sb->count, sizeof(int), 1, f);
    if (sb->count < 0 || sb->count > SCOREBOARD_MAX)
        sb->count = 0;
    else
        fread(sb->entries, sizeof(ScoreEntry), sb->count, f);

    fclose(f);
}

//Сохранениеивфайл
void Scoreboard_Save(const Scoreboard* sb) {
    FILE* f = fopen(SCOREBOARD_FILE, "wb");
    if (!f) return;

    fwrite(&sb->count, sizeof(int), 1, f);
    fwrite(sb->entries, sizeof(ScoreEntry), sb->count, f);
    fclose(f);
}

// Вставкарезультата
int Scoreboard_Submit(Scoreboard* sb, const char* name, int score,
bool victory) {
    // Новаязапись
    ScoreEntry ne;
    strncpy(ne.name, name, PLAYER_NAME_LEN - 1);
```

```

    ne.name[PLAYER_NAME_LEN - 1] = '\0';
    ne.score = score;
    ne.victory = victory;

    // Найти позицию для вставки (по убыванию очков)
    int pos = sb->count;
    for (int i = 0; i < sb->count; i++) {
        if (score > sb->entries[i].score) { pos = i; break; }
    }

    // Невошёлв TOP-10
    if (pos >= SCOREBOARD_MAX) return -1;

    // Сдвигаемхвоствправо
    int newCount = sb->count < SCOREBOARD_MAX ? sb->count + 1 :
SCOREBOARD_MAX;
    for (int i = newCount - 1; i > pos; i--)
        sb->entries[i] = sb->entries[i - 1];

    sb->entries[pos] = ne;
    sb->count = newCount;

    Scoreboard_Save(sb);
    return pos;
}

// Отрисовкатаблицы
void Scoreboard_Draw(const Scoreboard* sb, int highlightIdx) {
    const int panelW = 500;
    const int panelH = 60 + SCOREBOARD_MAX * 36 + 20;
    const int panelX = SCREEN_W / 2 - panelW / 2;
    const int panelY = SCREEN_H - 500;

    // Фонпанели
    DrawRectangle(panelX, panelY, panelW, panelH, (Color) { 0, 0, 30,
220 });
    DrawRectangleLines(panelX, panelY, panelW, panelH, (Color) { 100,
180, 255, 200 });

```

```

// Заголовок
DrawText("      TOP 10      ",
        panelX + panelW / 2 - MeasureText("      TOP 10      ", 24) / 2,
        panelY + 12, 24, GOLD);

if (sb->count == 0) {
    DrawText("There are no records", panelX + panelW / 2 -
60, panelY + 56, 20, GRAY);
    return;
}

for (int i = 0; i < sb->count; i++) {
    int rowY = panelY + 52 + i * 36;
    bool hi = (i == highlightIdx);

// Подсветка новой записи
    if (hi)
DrawRectangle(panelX + 4, rowY - 2, panelW - 8, 32,
        (Color) {
            255, 220, 0, 40
        });

    Color numCol = hi ? GOLD : GRAY;
    Color nameCol = hi ? YELLOW : WHITE;
    Color scCol = hi ? GOLD : LIGHTGRAY;

// Место
    DrawText(TextFormat("%2d.", i + 1), panelX + 12, rowY, 22,
numCol);

// Победа / поражение
    const char* badge = sb->entries[i].victory ? "[V]" : "[X]";
    Color badgeCol = sb->entries[i].victory ? GREEN : RED;
    DrawText(badge, panelX + 52, rowY, 20, badgeCol);

// Имя
    DrawText(sb->entries[i].name, panelX + 100, rowY, 22,
nameCol);

```

```
// Очки (выровнены по правому краю)
const char* scStr = TextFormat("%d", sb->entries[i].score);
    int scW = MeasureText(scStr, 22);
    DrawText(scStr, panelX + panelW - scW - 16, rowY, 22, scCol);
}
}
```

## Файл render\_utils.c

```
#include "game.h"

void DrawShadow(Texture2D tex, Rectangle src, Rectangle dst,
                Vector2 origin, float rotation,
                int layers, float step, int alpha)
{
    for (int i = layers; i >= 1; i--) {
        float t = (float)i / layers;
        int a = (int)(alpha * t * 0.6f);
        Rectangle sdst = {
            dst.x + step * i,
            dst.y + step * i,
            dst.width,
            dst.height
        };
        DrawTexturePro(tex, src, sdst, origin, rotation,
            (Color) {
                0, 0, 0, a
            });
    }
}
```

## Приложение В. Результаты работы программы

На рисунке 11 представлено главное меню программы. Пункты меню отображаются по центру экрана на фоне движущегося игрового поля. Выбранный пункт подсвечивается жёлтым цветом с мигающим маркером.

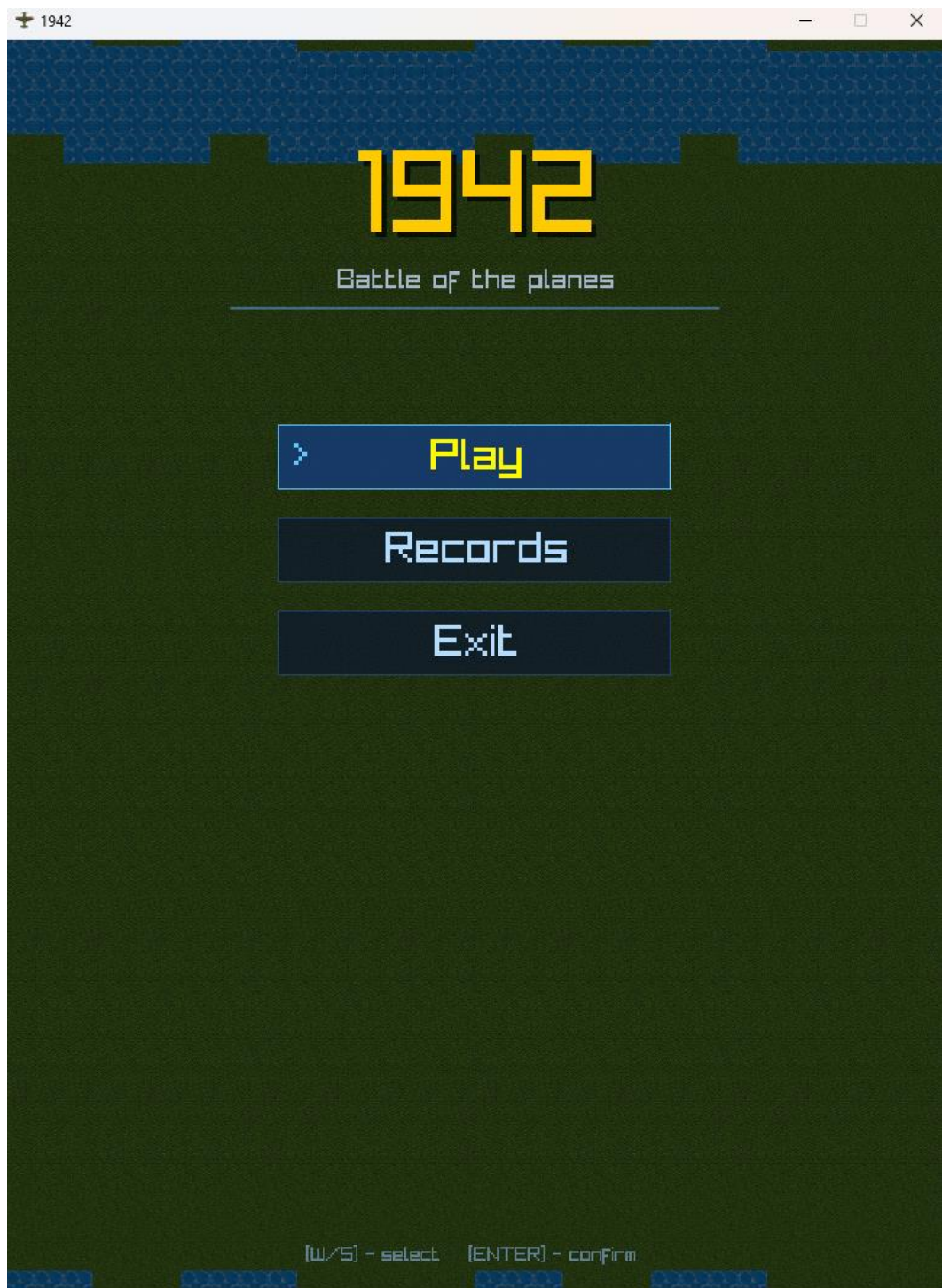


Рисунок 11 — Главное меню

На рисунке 12 показан игровой процесс с отображением врага типа «Босс». В верхней части экрана видны HP-бар босса, счёт игрока, иконки жизней и таймер.

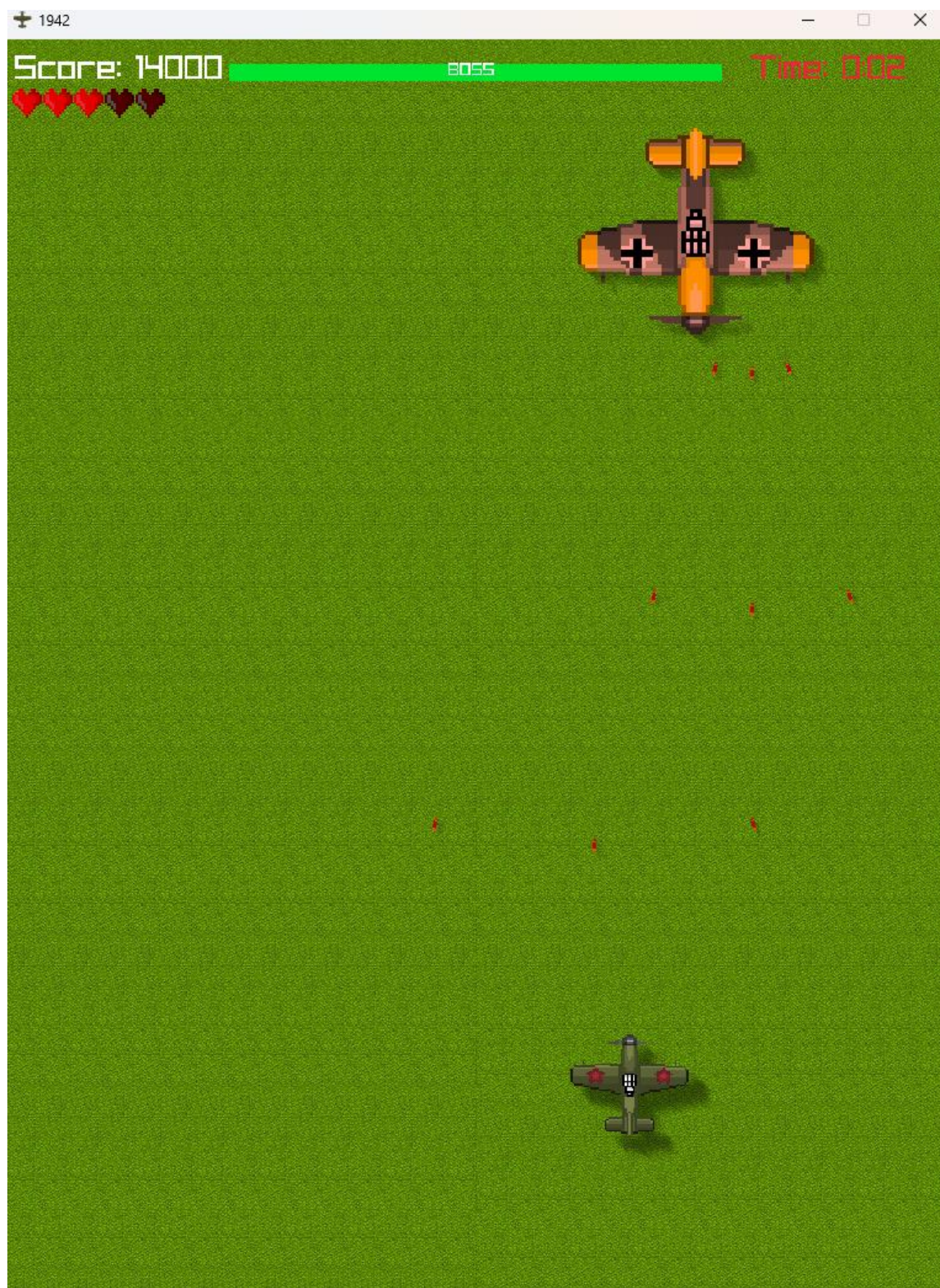


Рисунок 12 — Финальный босс с HP-баром

На рисунке 13 показан игровой процесс: самолёт игрока в нижней части экрана, враги типа «Камикадзе» и «Файтер», пули. В верхней части экрана отображается HUD.

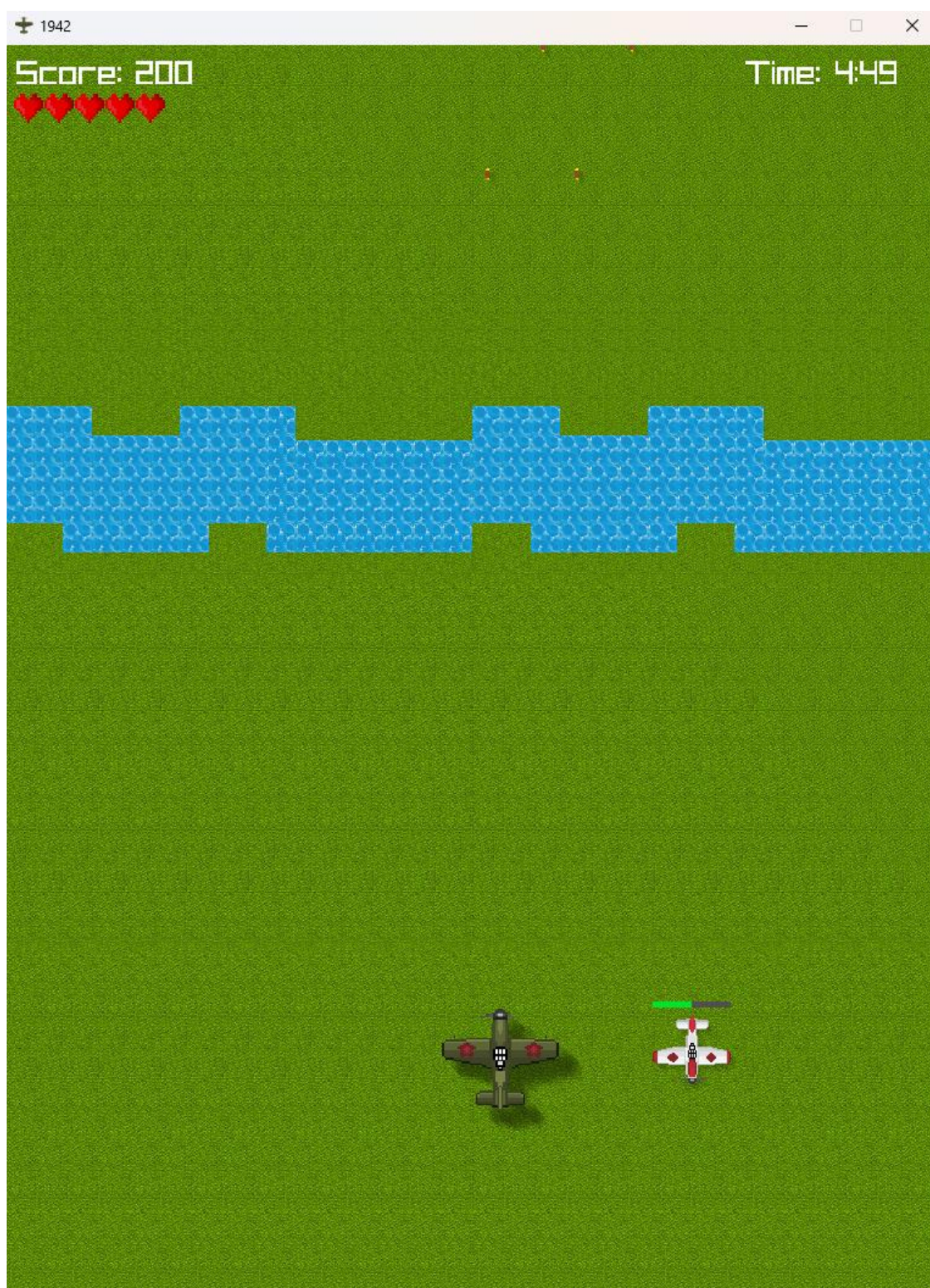


Рисунок 13 — Игровой процесс (HUD, враги, пули)

На рисунке 14 показан экран окончания игры с полем ввода имени, таблицей рекордов TOP-10 и подсветкой новой записи.



Рисунок 14 — Экран окончания игры и таблица рекордов