

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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по курсу «Программирование»
к курсовой работе

на тему «Разработка программы сложной структуры методом нисходящего
программирования. Игра «Бешеный динозавр»
на тему «Разработка программы сложной структуры методом нисходящего
программирования. Игра «Бешеный динозавр»

*оформлено
08.06.2026
Г. Генералова*

Выполнил: ст. гр. 25BVB1
Бардычев А.А.

Проверил: к.т.н. доцент

Выполнил: ст. гр. 25BVB1
Генералова А.А.
Бардычев А.А.

Проверил: к.т.н. доцент
Генералова А.А.

Пенза 2026

Пенза 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

М.А. Митрохин

« 18 » февраля 2026 г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Бардачеву Артёму Александровичу Группа 25 ВВВ1

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

Исходные данные (технические требования) на проектирование
игра "Бешеный динозавр"
Генерацию карты препятствий производит
компьютер. Игрок пытается преодолеть эту карту
и дойти до финиша
В файл записывается и сохраняется таблица
рекордов.

Обязательные требования к программе:

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

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

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

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания "18" февраля 2026г.

Дата защиты проекта "03" июня

Руководитель Г. Киселев А.А.

Задание получил "18" февраля 2026г.

Студент Богданов А.А.

Содержание

Введение.....	5
1 Постановка задачи.....	6
2 Выбор решения.....	6
3 Описание разработки программы.....	8
4 Отладка и тестирование.....	9
5 Описание программы.....	10
5.1 Схема функции main	10
5.2 Описание модулей.....	11
6 Руководство пользователя.....	14
6.1 Заставка	14
6.2 Правила игры	14
6.3 Главное меню.....	15
6.4 Игровой процесс.....	16
6.5 Подсчёт монет	16
6.6 Завершение игры и сохранение рекордов	17
Заключение	19
Список используемых материалов	20
Приложение А	21
Листинг программы	21
Файл main.py.....	21
Файл config.py.....	25
Файл game_objects.py.....	26
Файл records.py	27
Файл asm_collision.py.....	28
Файл collision.asm.....	29

Введение

В современном мире компьютерные игры занимают значительное место в сфере развлечений и даже профессиональной деятельности. Программирование подобных игр на языке Python подробно рассмотрено в работах Лутца [1], а также в учебнике Доусона [2]. Одной из таких игр можно считать популярную аркаду Динозаврик гугл хром за счёт того, что она развивает реакцию и умение принимать решения в условиях ограниченного времени. Эта игра представляет интерес с точки зрения алгоритмизации и программирования. Изучение принципов работы этой игры позволяет углубить знания в области разработки программного обеспечения, обработки пользовательского ввода, реализации сложной игровой логики с использованием объектно-ориентированного подхода и графической библиотеки Pygame.

Актуальность работы заключается в том, что реализация игры «Бешеный динозавр» охватывает целый ряд фундаментальных тем программирования: обработку пользовательского ввода, обновление игрового состояния в реальном времени, проверку столкновений объектов, генерацию случайных препятствий и сбор предметов, а также сохранение и загрузку рекордов.

В данной работе используется язык программирования Python с библиотекой Pygame, потому что этот язык предоставляет высокоуровневые абстракции для разработки игр, а Pygame даёт удобный доступ к графике и обработке событий. Также в работе применяется ассемблерная вставка (в отдельном модуле) для оптимизации проверки столкновений — наиболее критичной по производительности части игры.

1 Постановка задачи

Целью данной работы является разработка графической игры «Бешеный динозавр» с использованием языка программирования Python и библиотеки Pygame, а также с применением ассемблерной оптимизации для проверки столкновений. В ходе выполнения работы необходимо реализовать следующий функционал:

1. Графическое окно с динамически обновляемым изображением
2. Управление персонажем (динозавром) с возможностью прыжка
3. Генерацию препятствий (кактусов) случайной высоты
4. Генерацию собираемых предметов (монет) на разных высотах
5. Проверку столкновений динозавра с препятствиями и монетами
6. Систему подсчёта очков (монет) с прогрессивным усложнением
7. Увеличение скорости игры при наборе очков
8. Таблицу рекордов с сохранением в файл (лучшие 5 результатов)
9. Режим отладки (F3) для визуализации хитбоксов объектов

Во время реализации проекта должны быть использованы сложные типы данных — классы, списки, файлы, алгоритмы обработки ввода и вывода и принципы модульного программирования.

Сохранение результатов игр в текстовый файл records.txt.

Режим работы видеосистемы — графический (Pygame, 1000×600 пикселей).

Устройство ввода информации — клавиатура.

Пользовательский интерфейс должен быть построен на основе графического интерфейса с меню и внутриигровым HUD.

Наличие заставки (начальное меню с правилами и рекордами).

2 Выбор решения

Для разработки выбрана среда PyCharm с интерпретатором Python 3.11 и библиотекой Pygame. Она предоставляет все необходимые инструменты для

комфортной и эффективной разработки игры «Бешеный динозавр», делая процесс написания кода, отладки и оптимизации максимально удобным.

Проект структурирован с использованием модулей, что позволяет логически разделить код на функциональные блоки, которые легко сопровождать. Схема данных приведена на рисунке 1.

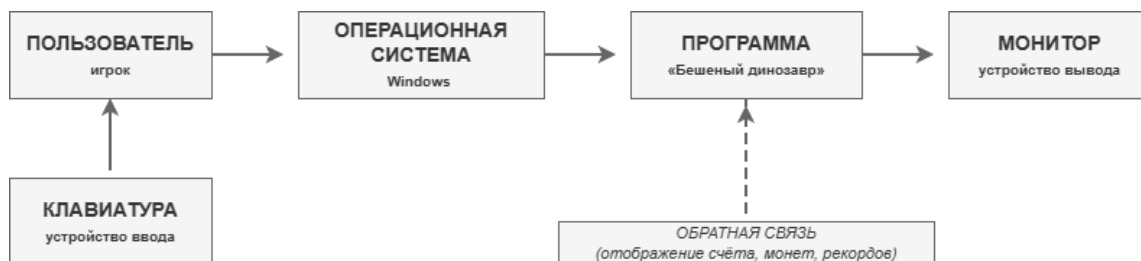


Рисунок 1 – Схема данных

Python также подходит для работы с ассемблерными вставками через `ctypes` и отдельно скомпилированные библиотеки, что позволяет интегрировать низкоуровневый код в высокоуровневое приложение.

Главным преимуществом является возможность комбинирования Python для быстрой разработки игровой логики и ассемблера для оптимизации критических участков кода (проверки столкновений). Ассемблерная вставка реализована в отдельном файле `collision.asm` и компилируется в динамическую библиотеку (`collision.dll/.so/.dylib`), которая загружается через `ctypes`.

Данные о рекордах сохраняются в текстовый файл `records.txt` через функции `load_records()` / `save_records()`. По завершению игры в файл записываются лучшие результаты (до 5 записей) в порядке убывания.

3 Описание разработки программы

Для реализации был выбран нисходящий метод разработки, потому что он позволяет сначала определить общую структуру программы, а затем постепенно детализировать её компоненты.

При запуске программы на экран выводится приветственная заставка с правилами игры и текущим рекордом:

- **Цель:** собрать указанное количество монет (начинается с 10)
- **Управление:** ПРОБЕЛ / СТРЕЛКА ВВЕРХ / W — прыжок
- **Усложнение:** после каждой победы цель увеличивается на 5 монет

Программа разделена на 6 модулей, потому что это оптимальный вариант, который позволяет изменять одну из частей, не затрагивая остальные, а также соответствует принципам структурного программирования [3]:

1. `config.py` — все константы игры (размеры окна, цвета, физика)
2. `game_objects.py` — классы `Dino`, `Obstacle`, `Coin`
3. `main.py` — главный модуль: игровой цикл, меню, отрисовка
4. `records.py` — работа с таблицей рекордов (загрузка/сохранение)
5. `asm_collision.py` — загрузчик ассемблерной библиотеки
6. `collision.asm` — ассемблерная реализация проверки столкновений

4 Отладка и тестирование

В качестве среды разработки была выбрана программа PyCharm. Она оптимальна как для разработки на языке Python, так и для отладки программы благодаря мощному функционалу и удобству работы.

Точки останова позволяют исследовать состояние программы, чтобы проверить, как ведёт себя приложение. На рисунке 2 показан пример использования точек останова.

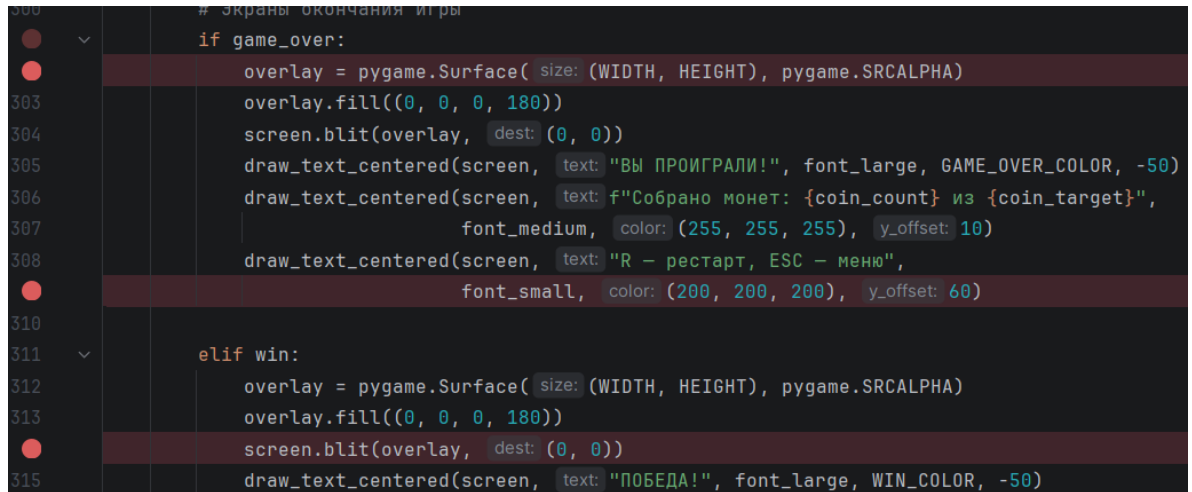


Рисунок 2 – Использование точек останова

Тестирование проводилось в рабочем процессе, в процессе разработки и после завершения написания программы. Были протестированы:

1. Корректность проверки столкновений
2. Механика прыжка и гравитация
3. Генерация препятствий и монет
4. Система рекордов
5. Режим отладки (F3) — визуализация хитбоксов

5 Описание программы

5.1 Схема функции main

На рисунке 3 представлена схема функции main.

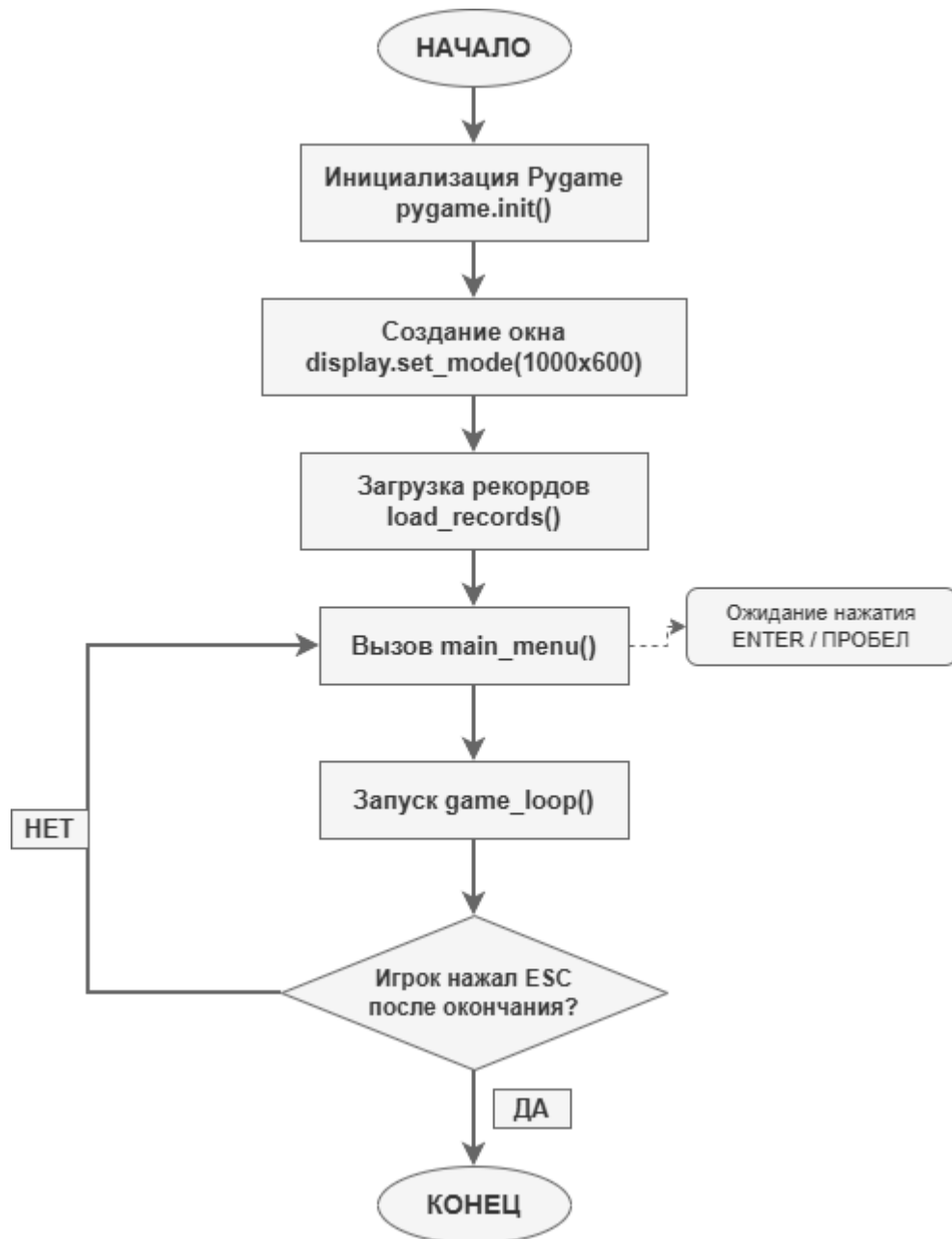


Рисунок 3 – Схема функции main

На схеме представлена последовательность действий при запуске программы: инициализация Pygame, загрузка рекордов, затем бесконечный игровой цикл с обработкой событий, обновлением игрового состояния, отрисовкой и проверкой условий победы/поражения.

5.2 Описание модулей

Проект состоит из следующих модулей:

1. **config.py** — конфигурационный файл, содержит:
 - Размеры окна (1000×600)
 - Параметры физики (гравитация, сила прыжка)
 - Размеры объектов (динозавр 50×60, кактус 25×40–80, монета радиусом 15)
 - Цвета в формате RGB
 - Параметры генерации объектов
2. **game_objects.py** — классы игровых объектов:
 - Dino — динозавр: координаты, размер, скорость прыжка, методы `update()` и `draw()`
 - Obstacle — кактус: случайная высота, движение влево, отрисовка с колючками
 - Coin — монета: случайная высота из трёх фиксированных, отрисовка с бликом
3. **main.py** — главный модуль:
 - `check_collision_rect()` — проверка столкновений (можно переключить на ассемблерную)
 - `draw_background()` — отрисовка неба и земли
 - `main_menu()` — начальное меню
 - `game_loop()` — основной игровой цикл
 - Обработка событий (прыжок, F3 — отладка, R — рестарт, ESC — выход)

4. **records.py** — работа с рекордами:

- `load_records()` — загрузка из файла `records.txt`
- `save_records()` — сохранение в файл
- `add_record()` — добавление нового результата
- `get_best_score()` — получение лучшего результата

5. **asm_collision.py** — загрузчик ассемблерной библиотеки:

- `load_asm_collision()` — загрузка `collision.dll/.so/.dylib` через `ctypes`
- Определение сигнатуры функции для разных ОС (Windows x64 / System V)

6. **collision.asm** — ассемблерная реализация (NASM, x86-64):

- Поддержка двух соглашений вызова (Windows x64 и System V)
- Проверка пересечения двух прямоугольников по осям X и Y
- Возврат 1 при столкновении, 0 иначе

Ассемблерная вставка демонстрирует низкоуровневую проверку столкновений [3]:

```
; Проверка X: x1 + w1 <= x2 ?
mov eax, edi
add eax, edx
cmp eax, r8d
jle .no_collision

; Проверка X: x2 + w2 <= x1 ?
mov eax, r8d
add eax, [rbp+16]
cmp eax, edi
jle .no_collision

; Проверка Y: y1 + h1 <= y2 ?
mov eax, esi
add eax, ecx
cmp eax, r9d
```

```
jle .no_collision

; Проверка Y:  $y_2 + h_2 \leq y_1$  ?
mov eax, r9d
add eax, [rbp+24]
cmp eax, esi
jle .no_collision

; Столкновение есть
mov eax, 1
```

Нисходящий метод проектирования подробно описан у Лутца [1], а фундаментальные принципы работы ассемблера изложены в классическом учебнике Ирвина [4].

6 Руководство пользователя

Программа main.ru является графической игрой, имитирующей механику популярной аркады.

6.1 Заставка

На рисунке 4 представлена игровая заставка, которая появляется при запуске приложения.

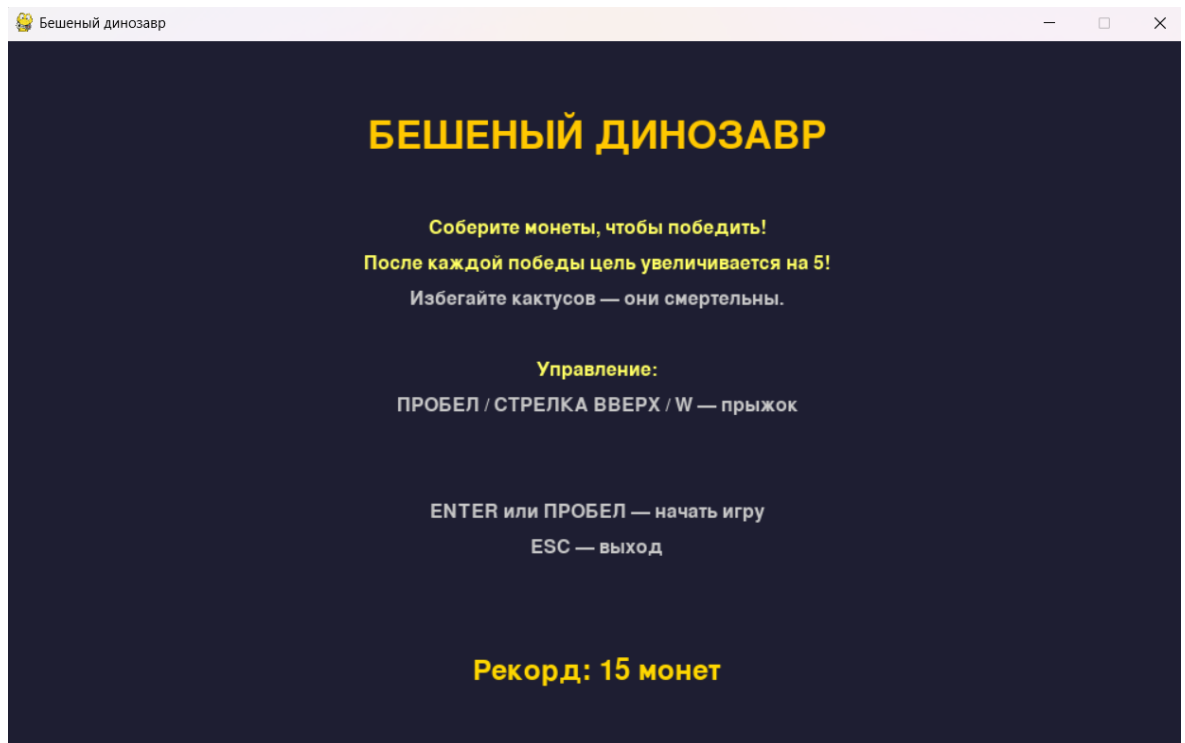


Рисунок 4 – Заставка

Меню оформлено с использованием цветного текста и интуитивно понятными подсказками для наглядности.

6.2 Правила игры

На рисунке 5 представлены правила игры.

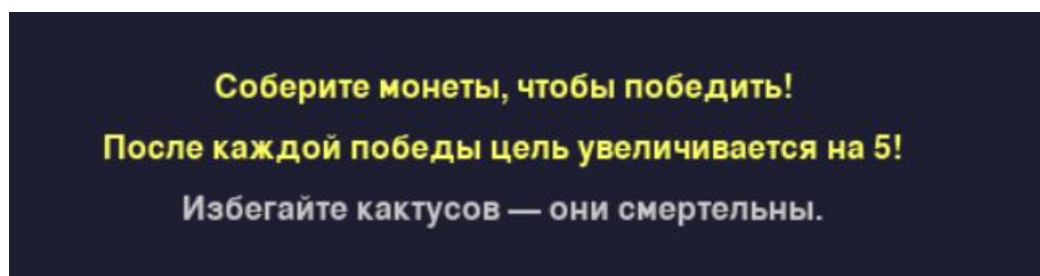


Рисунок 5 – Правила игры

Правила представлены в сжатом виде, чтобы не перегружать пользователя, но при этом дать полное понимание механики: сбор монет, избегание кактусов, увеличение цели после победы.

6.3 Главное меню

На рисунке 6 представлено главное меню игры, которое содержит:

1. Название игры
2. Правила игры
3. Управление (ПРОБЕЛ / ↑ / W — прыжок)
4. Текущий рекорд
5. Приглашение начать игру (ENTER или ПРОБЕЛ)



Рисунок 6 – главное меню

6.4 Игровой процесс

После запуска игры игрок управляет динозавром, который бежит по земле. На рисунке 7 представлен игровой процесс.

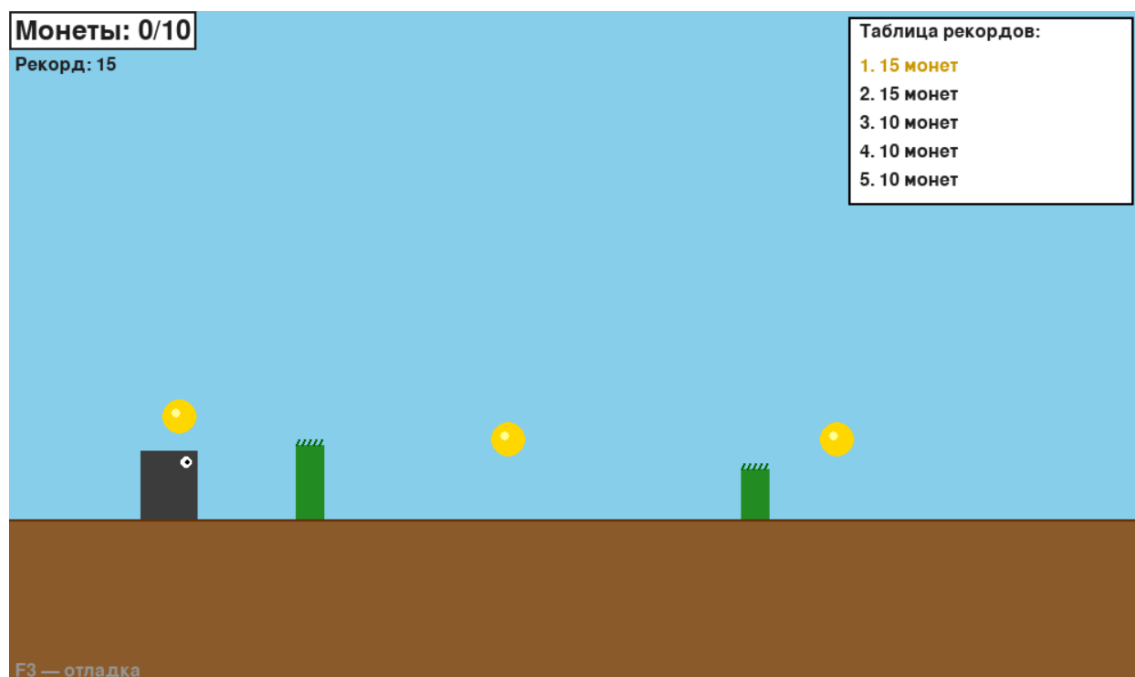


Рисунок 7 – игра

На экране отображаются:

- Динозавр (тёмно-серый прямоугольник с глазом)
- Кактусы (зелёные прямоугольники с колючками)
- Монеты (золотые круги)
- Счётчик монет (текущее/целевое количество)
- Текущий рекорд
- Таблица рекордов (справа)

6.5 Подсчёт монет

Подсчёт автоматический. За каждую собранную монету счёт увеличивается на 1. При достижении целевого количества (начинается с 10) наступает победа, и цель увеличивается на 5 монет для следующей игры.

На рисунке 8 показан режим отладки (F3), в котором видны хитбоксы всех объектов (красные рамки для динозавра и кактусов, зелёные — для монет).

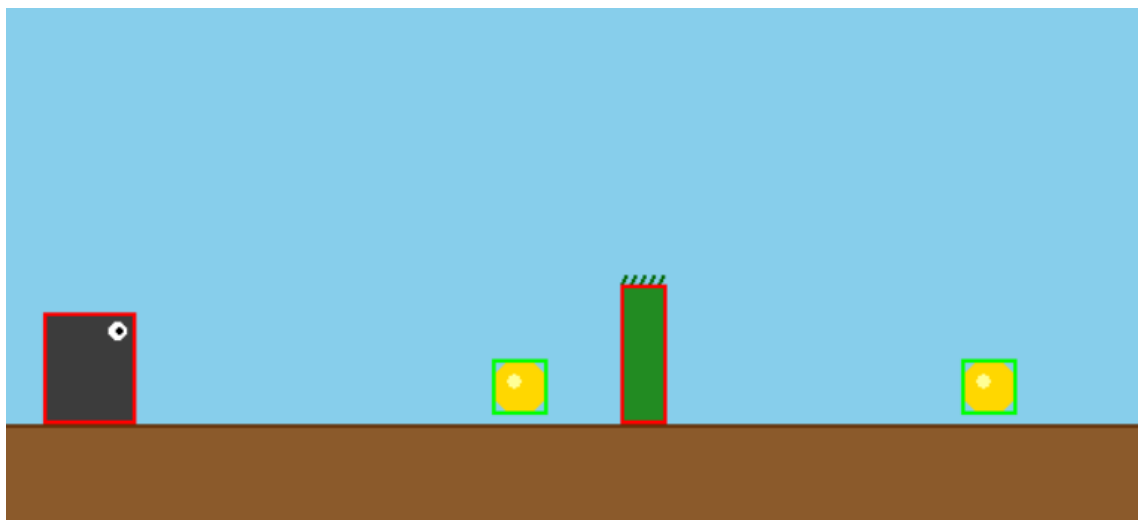


Рисунок 8 – режим отладки

6.6 Завершение игры и сохранение рекордов

При столкновении с кактусом игра заканчивается проигрышем (рис. 9).

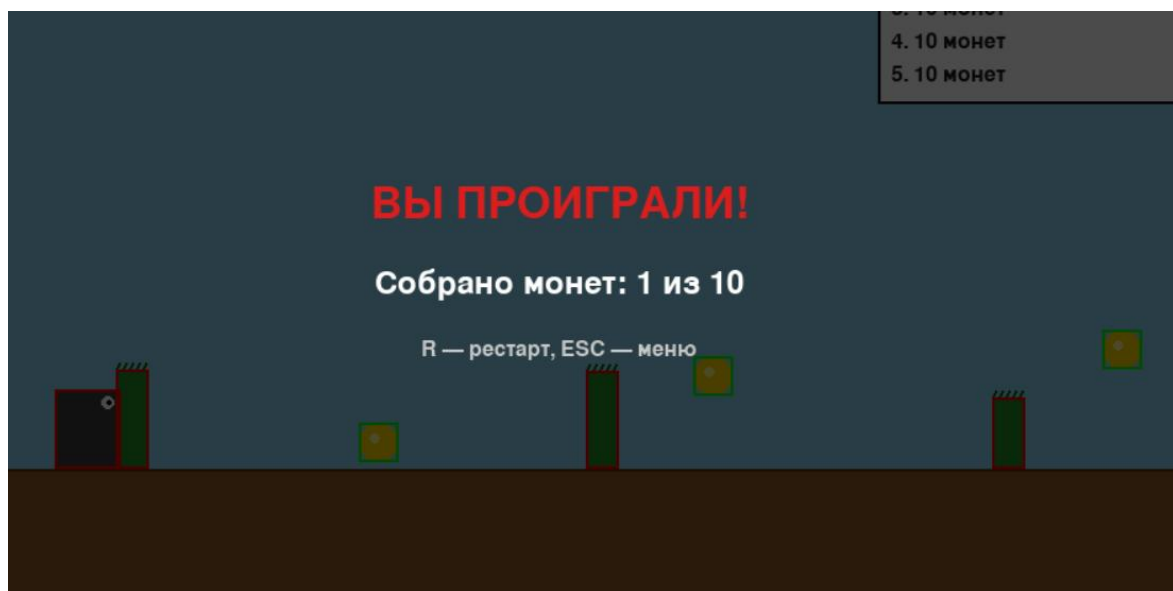


Рисунок 9 – экран проигрыша

При достижении целевого количества монет игра заканчивается победой (рис. 10)

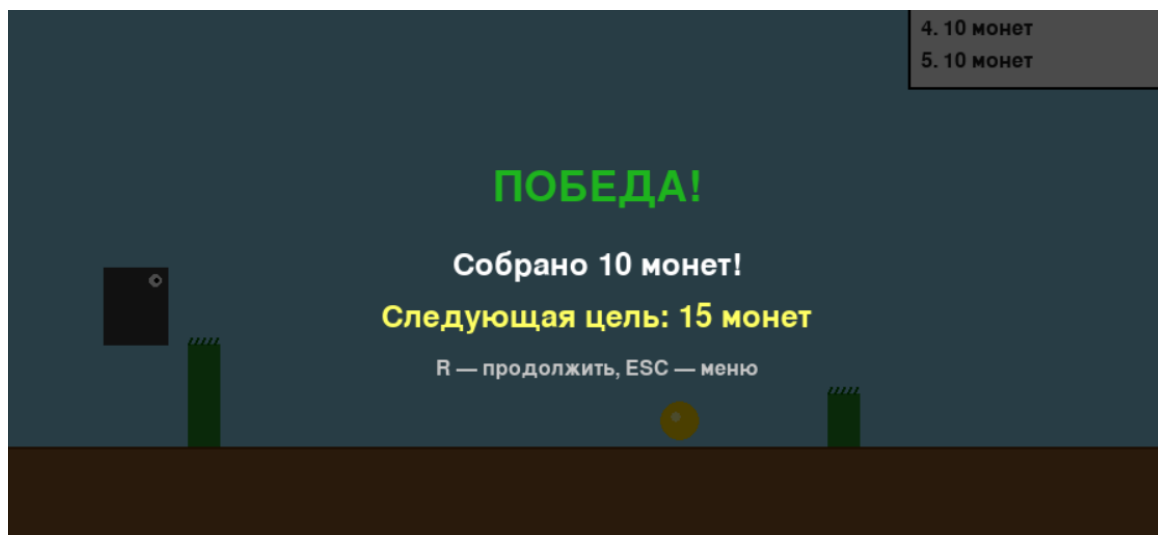


Рисунок 10 — экран победы

После каждой игры (победы или поражения) результат заносится в таблицу рекордов и сохраняется в файл records.txt (рис. 11).

```

15
15
10
10
10
  
```

Рисунок 11 — файл records.txt с рекордами

Таблицу рекордов можно посмотреть в правой части игрового экрана (рис. 12).

Таблица рекордов:	
1.	15 монет
2.	15 монет
3.	10 монет
4.	10 монет
5.	10 монет

рисунок 12 — таблица рекордов в игре

Программа автоматически обновляет эти данные после каждой игры и загружает их при запуске. Это позволяет сохранять статистику между сеансами игры. Таким образом, пользователь может отслеживать свои лучшие результаты.

Заключение

В ходе выполнения курсовой работы была разработана графическая игра «Бешеный динозавр» на языке Python с использованием библиотеки Pygame и с возможностью использования ассемблерной оптимизации для проверки столкновений.

Основные достижения:

1. Реализована полная логика аркадной игры «ранер» с динамическим усложнением.
2. Применено модульное программирование — это разделение на `config`, `game_objects`, `main`, `records`, `asm_collision`.
3. Использованы классы для представления игровых объектов, списки для хранения динамических объектов и файловый ввод/вывод для сохранения рекордов.
4. В критический алгоритм (проверка столкновений прямоугольников) реализована альтернативная ассемблерная версия для демонстрации низкоуровневых возможностей и оптимизации.
5. Создано удобное графическое меню, таблица рекордов и режим отладки для визуализации хитбоксов.

Проект показывает эффективность языка Python и библиотеки Pygame для написания игровой логики. Программа успешно реализует все основные функции игры «Бешеный динозавр». Приобретённые в ходе выполнения работы навыки имеют важное значение для дальнейшего профессионального роста в области программирования.

Список используемых материалов

1. Лутц М. Изучаем Python : пер. с англ. / М. Лутц. — 5-е изд. — Москва : Диалектика, 2019. — 1464 с.
2. Доусон М. Програмируем на Python : пер. с англ. / М. Доусон. — СПб. : Питер, 2019 — 464 с.
3. Прата, С. Язык программирования C++. Лекции и упражнения : пер. с англ. / С. Прата. — 6-е изд. — Москва : Диалектика, 2019. — 896 с.
4. Ирвин К. Язык ассемблера для процессоров Intel : пер. с англ. / К. Ирвин. — 7-е изд. — М. : Вильямс, 2017. — 912 с.
5. Зубков, С. В. Ассемблер для процессоров Intel x86 : учебное пособие / С. В. Зубков. — Москва : ДМК Пресс, 2020 — 412 с.
6. Подбельский В. В. Язык Си++ : учеб. пособие / В. В. Подбельский. — 6-е изд., перераб. и доп. — М. : Финансы и статистика, 2020. — 600 с.
7. Маккинак М. Pygame. Создание игр на Python : учеб. пособие / М. Маккинак. — М. : ДМК Пресс, 2018. — 256 с.

Приложение А

Листинг программы

Файл main.py

```
# main.py
import pygame
import sys
import random
import os

from config import *
from game_objects import Dino, Obstacle, Coin
from records import load_records, add_record, get_best_score, MAX_RECORDS

# =====
# Инициализация Pygame
# =====
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Бешеный динозавр")
clock = pygame.time.Clock()

# Шрифты
font_large = pygame.font.Font(None, 48)
font_medium = pygame.font.Font(None, 36)
font_small = pygame.font.Font(None, 24)

# =====
# Функция столкновений (Python-версия)
# =====
def check_collision_rect(x1, y1, w1, h1, x2, y2, w2, h2):
    """
    Проверяет пересечение двух прямоугольников.
    Возвращает True если пересекаются, False если нет.
    """
    if x1 + w1 <= x2:
        return False
    if x2 + w2 <= x1:
        return False
    if y1 + h1 <= y2:
        return False
    if y2 + h2 <= y1:
        return False
    return True

# =====
# Вспомогательные функции отрисовки
# =====
def draw_background(surf):
    surf.fill(SKY_COLOR)
    pygame.draw.rect(surf, GROUND_COLOR, (0, GROUND_Y, WIDTH, HEIGHT - GROUND_Y))
    pygame.draw.line(surf, (100, 50, 10), (0, GROUND_Y), (WIDTH, GROUND_Y), 2)

def draw_text_centered(surf, text, font, color, y_offset=0):
```

```

text_surf = font.render(text, True, color)
x = WIDTH // 2 - text_surf.get_width() // 2
y = HEIGHT // 2 - text_surf.get_height() // 2 + y_offset
surf.blit(text_surf, (x, y))

# =====
# Начальное меню
# =====
def main_menu():
    """Экран начального меню."""
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN or event.key == pygame.K_SPACE:
                    return
                if event.key == pygame.K_ESCAPE:
                    pygame.quit()
                    sys.exit()

    screen.fill((30, 30, 50))

    title = font_large.render("БЕШЕНЫЙ ДИНОЗАВР", True, (255, 200, 0))
    screen.blit(title, (WIDTH // 2 - title.get_width() // 2, 60))

    rules = [
        "Соберите монеты, чтобы победить!",
        "После каждой победы цель увеличивается на 5!",
        "Избегайте кактусов – они смертельны.",
        "",
        "Управление:",
        "ПРОБЕЛ / СТРЕЛКА ВВЕРХ / W – прыжок",
        "",
        "",
        "ENTER или ПРОБЕЛ – начать игру",
        "ESC – выход",
    ]

    for i, line in enumerate(rules):
        if line.startswith("Управление:") or line.startswith("Соберите") or
line.startswith("После"):
            color = (255, 255, 100)
        elif line == "":
            continue
        else:
            color = (200, 200, 200)
        text = font_small.render(line, True, color)
        screen.blit(text, (WIDTH // 2 - text.get_width() // 2, 150 + i * 30))

    best = get_best_score()
    if best > 0:
        best_text = font_medium.render(f"Рекорд: {best} монет", True, (255, 215, 0))
        screen.blit(best_text, (WIDTH // 2 - best_text.get_width() // 2, HEIGHT -
80))

    pygame.display.flip()
    clock.tick(FPS)

# =====
# Основной игровой цикл
# =====
def game_loop(coin_target=COIN_TARGET):
    dino = Dino()
    obstacles = []
    coins = []

```

```

coin_count = 0
speed = OBSTACLE_BASE_SPEED

spawn_timer_obstacle = 0
spawn_timer_coin = 0

debug_mode = False

running = True
game_over = False
win = False

while running:
    dt = clock.tick(FPS)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.KEYDOWN:
            if event.key in (pygame.K_SPACE, pygame.K_UP, pygame.K_w):
                if not game_over and not win:
                    dino.jump()

            if event.key == pygame.K_F3:
                debug_mode = not debug_mode
                print(f"[ОТЛАДКА] Режим отладки: {'ВКЛ' if debug_mode else 'ВЫКЛ'}")

            if event.key == pygame.K_r:
                if game_over or win:
                    if win:
                        return game_loop(coin_target + 5)
                    else:
                        return game_loop(coin_target)

            if event.key == pygame.K_ESCAPE:
                if game_over or win:
                    return

    if not game_over and not win:
        dino.update()

        spawn_timer_obstacle += 1
        if spawn_timer_obstacle > random.randint(50, 100):
            obstacles.append(Obstacle(speed))
            spawn_timer_obstacle = 0

        spawn_timer_coin += 1
        if spawn_timer_coin > random.randint(40, 80):
            coins.append(Coin(speed))
            spawn_timer_coin = 0

        # Проверка столкновений с препятствиями
        for obs in obstacles[:]:
            obs.update()

            if check_collision_rect(
                dino.x, dino.y, dino.width, dino.height,
                obs.x, obs.y, obs.width, obs.height
            ):
                game_over = True
                add_record(coin_count)
                break

            if obs.off_screen():
                obstacles.remove(obs)

        # Проверка сбора монет

```

```

    for coin in coins[:]:
        coin.update()

        coin_left = coin.x - coin.radius
        coin_top = coin.y - coin.radius
        coin_size = coin.radius * 2

        if check_collision_rect(
            dino.x, dino.y, dino.width, dino.height,
            coin_left, coin_top, coin_size, coin_size
        ):
            coins.remove(coin)
            coin_count += 1

            if coin_count >= coin_target:
                win = True
                add_record(coin_count)

            elif coin.off_screen():
                coins.remove(coin)

    speed = OBSTACLE_BASE_SPEED + coin_count * 0.3

# Отрисовка
draw_background(screen)

dino.draw(screen)
for obs in obstacles:
    obs.draw(screen)
for coin in coins:
    coin.draw(screen)

if debug_mode:
    pygame.draw.rect(screen, (255, 0, 0),
                      (dino.x, dino.y, dino.width, dino.height), 2)
    for obs in obstacles:
        pygame.draw.rect(screen, (255, 0, 0),
                          (obs.x, obs.y, obs.width, obs.height), 2)
    for coin in coins:
        pygame.draw.rect(screen, (0, 255, 0),
                          (coin.x - coin.radius, coin.y - coin.radius,
                           coin.radius * 2, coin.radius * 2), 2)

score_text = font_medium.render(
    f"Монеты: {coin_count}/{coin_target}", True, TEXT_COLOR
)
score_bg = pygame.Rect(5, 5, score_text.get_width() + 10,
score_text.get_height() + 10)
pygame.draw.rect(screen, (255, 255, 255), score_bg)
pygame.draw.rect(screen, TEXT_COLOR, score_bg, 2)
screen.blit(score_text, (10, 10))

best = get_best_score()
best_text = font_small.render(f"Рекорд: {best}", True, TEXT_COLOR)
screen.blit(best_text, (10, score_text.get_height() + 20))

hint_text = font_small.render("F3 - отладка", True, (150, 150, 150))
screen.blit(hint_text, (10, HEIGHT - 25))

records_list = load_records()
if records_list:
    panel_rect = pygame.Rect(WIDTH - 260, 10, 250, 40 + min(MAX_RECORDS,
len(records_list)) * 25)
    pygame.draw.rect(screen, (255, 255, 255), panel_rect)
    pygame.draw.rect(screen, (0, 0, 0), panel_rect, 2)
    title = font_small.render("Таблица рекордов:", True, TEXT_COLOR)
    screen.blit(title, (WIDTH - 250, 15))
    for i, rec in enumerate(records_list[:MAX_RECORDS]):
        color = (200, 150, 0) if i == 0 else TEXT_COLOR

```

```

        text = font_small.render(f"{i + 1}. {rec} монет", True, color)
        screen.blit(text, (WIDTH - 250, 45 + i * 25))

    if game_over:
        overlay = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)
        overlay.fill((0, 0, 0, 180))
        screen.blit(overlay, (0, 0))
        draw_text_centered(screen, "ВЫ ПРОИГРАЛИ!", font_large, GAME_OVER_COLOR, -
50)
        draw_text_centered(screen, f"Собрано монет: {coin_count} из {coin_target}",
                             font_medium, (255, 255, 255), 10)
        draw_text_centered(screen, "R – рестарт, ESC – меню",
                             font_small, (200, 200, 200), 60)

    elif win:
        overlay = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)
        overlay.fill((0, 0, 0, 180))
        screen.blit(overlay, (0, 0))
        draw_text_centered(screen, "ПОБЕДА!", font_large, WIN_COLOR, -50)
        draw_text_centered(screen, f"Собрано {coin_count} монет!",
                             font_medium, (255, 255, 255), 10)
        draw_text_centered(screen, f"Следующая цель: {coin_target + 5} монет",
                             font_medium, (255, 255, 100), 50)
        draw_text_centered(screen, "R – продолжить, ESC – меню",
                             font_small, (200, 200, 200), 90)

    pygame.display.flip()

    return coin_count

# =====
# Точка входа
# =====
if __name__ == "__main__":
    print("=" * 60)
    print(" БЕШЕНЫЙ ДИНОЗАВР")
    print(f" Начальная цель: собрать {COIN_TARGET} монет")
    print(" После каждой победы цель увеличивается на 5!")
    print(" Управление: ПРОБЕЛ / ↑ / W – прыжок")
    print(" F3 – показать рамки столкновений")
    print("=" * 60)

    while True:
        main_menu()
        game_loop(COIN_TARGET)

```

Файл config.py

```

# config.py
WIDTH, HEIGHT = 1000, 600
GROUND_Y = 450
FPS = 60
GRAVITY = 0.8
JUMP_STRENGTH = -15
DINO_WIDTH = 50
DINO_HEIGHT = 60
DINO_X = 120
OBSTACLE_WIDTH = 25
OBSTACLE_MIN_H = 40
OBSTACLE_MAX_H = 80
OBSTACLE_BASE_SPEED = 6
COIN_RADIUS = 15
COIN_TARGET = 10
COIN_SPEED = 6
SKY_COLOR = (135, 206, 235)
GROUND_COLOR = (139, 90, 43)
GROUND_LINE_COLOR = (100, 60, 20)
DINO_COLOR = (60, 60, 60)

```

```

DINO_EYE_WHITE = (255, 255, 255)
DINO_EYE_BLACK = (0, 0, 0)
OBSTACLE_COLOR = (34, 139, 34)
OBSTACLE_SPIKE_COLOR = (0, 100, 0)
COIN_COLOR = (255, 215, 0)
COIN_BLING_COLOR = (255, 255, 150)
TEXT_COLOR = (30, 30, 30)
GAME_OVER_COLOR = (220, 30, 30)
WIN_COLOR = (30, 180, 30)

```

Файл game_objects.py

```

# game_objects.py
import pygame
import random
from config import *

class Dino:
    def __init__(self):
        self.x = DINO_X
        self.y = GROUND_Y - DINO_HEIGHT
        self.width = DINO_WIDTH
        self.height = DINO_HEIGHT
        self.vel_y = 0
        self.on_ground = True

    def jump(self):
        if self.on_ground:
            self.vel_y = JUMP_STRENGTH
            self.on_ground = False

    def update(self):
        self.y += self.vel_y
        self.vel_y += GRAVITY
        if self.y >= GROUND_Y - self.height:
            self.y = GROUND_Y - self.height
            self.vel_y = 0
            self.on_ground = True

    def draw(self, surf):
        rect = pygame.Rect(self.x, self.y, self.width, self.height)
        pygame.draw.rect(surf, DINO_COLOR, rect)
        eye_x = self.x + self.width - 10
        eye_y = self.y + 10
        pygame.draw.circle(surf, DINO_EYE_WHITE, (eye_x, eye_y), 5)
        pygame.draw.circle(surf, DINO_EYE_BLACK, (eye_x + 1, eye_y), 2)

class Obstacle:
    def __init__(self, speed):
        self.width = OBSTACLE_WIDTH
        self.height = random.randint(OBSTACLE_MIN_H, OBSTACLE_MAX_H)
        self.x = WIDTH
        self.y = GROUND_Y - self.height
        self.speed = speed

    def update(self):
        self.x -= self.speed

    def off_screen(self):
        return self.x + self.width < 0

    def draw(self, surf):
        rect = pygame.Rect(self.x, self.y, self.width, self.height)
        pygame.draw.rect(surf, OBSTACLE_COLOR, rect)
        for i in range(0, self.width, 5):
            pygame.draw.line(surf, OBSTACLE_SPIKE_COLOR,
                             (self.x + i, self.y),

```

```
(self.x + i + 2, self.y - 5), 2)
```

```
class Coin:
    def __init__(self, speed):
        self.radius = COIN_RADIUS
        self.x = WIDTH
        self.speed = speed

        # Все монеты на одной из трёх фиксированных высот
        heights = [
            GROUND_Y - self.radius - 5, # На земле
            GROUND_Y - DINO_HEIGHT - 10, # На уровне груди
            GROUND_Y - DINO_HEIGHT - 30, # Над головой
        ]
        self.y = random.choice(heights)

    def update(self):
        self.x -= self.speed

    def off_screen(self):
        return self.x + self.radius < 0

    def draw(self, surf):
        pygame.draw.circle(surf, COIN_COLOR, (self.x, self.y), self.radius)
        pygame.draw.circle(surf, COIN_BLING_COLOR, (self.x - 3, self.y - 3), 4)
```

Файл records.py

```
# records.py
import os

RECORDS_FILE = "records.txt"
MAX_RECORDS = 5

def load_records():
    """Загружает список рекордов из файла."""
    records = []
    if os.path.exists(RECORDS_FILE):
        try:
            with open(RECORDS_FILE, "r", encoding="utf-8") as f:
                for line in f:
                    line = line.strip()
                    if line.isdigit():
                        records.append(int(line))
        except Exception as e:
            print(f"Предупреждение: не удалось загрузить рекорды: {e}")
    return records

def save_records(records):
    """Сохраняет список рекордов в файл (только лучшие MAX_RECORDS результатов)."""
    records = sorted(records, reverse=True)[:MAX_RECORDS]
    try:
        with open(RECORDS_FILE, "w", encoding="utf-8") as f:
            for r in records:
                f.write(f"{r}\n")
    except Exception as e:
        print(f"Предупреждение: не удалось сохранить рекорды: {e}")

def add_record(score):
    """Добавляет новый результат в таблицу рекордов."""
    records = load_records()
    records.append(score)
    save_records(records)

def get_best_score():
    """Возвращает лучший результат или 0, если рекордов нет."""
    records = load_records()
    return max(records) if records else 0
```

Файл asm_collision.py

```
# asm_collision.py
import ctypes
import sys
import os

def load_asm_collision():
    """
    Загружает скомпилированную ассемблерную библиотеку (collision.dll /
    collision.so / collision.dylib)
    и возвращает функцию collision.

    Сигнатура функции:
    int collision(int x1, int y1, int w1, int h1, int x2, int y2, int w2, int
    h2)

    Возвращает 1, если прямоугольники пересекаются, иначе 0.
    """

    # Определяем сигнатуру функции для ctypes
    CollisionFunc = ctypes.CFUNCTYPE(
        ctypes.c_int, # возвращаемое значение
        ctypes.c_int, # x1
        ctypes.c_int, # y1
        ctypes.c_int, # w1
        ctypes.c_int, # h1
        ctypes.c_int, # x2
        ctypes.c_int, # y2
        ctypes.c_int, # w2
        ctypes.c_int, # h2
    )

    # Выбираем имя файла библиотеки в зависимости от ОС
    if sys.platform == "win32":
        lib_name = "collision.dll"
    elif sys.platform == "darwin":
        lib_name = "collision.dylib"
    else:
        # Linux и другие Unix-подобные
        lib_name = "collision.so"

    # Полный путь к библиотеке (в той же папке, что и этот файл)
    lib_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), lib_name)

    # Проверяем, существует ли файл
    if not os.path.exists(lib_path):
        raise FileNotFoundError(
            f"Не найдена ассемблерная библиотека: {lib_path}\n"
            f"Сначала скомпилируйте её:\n"
            f"  Windows (cmd.exe): compile.bat\n"
            f"  Linux/macOS:      bash compile.sh"
        )

    # Загружаем библиотеку
    try:
        collision_lib = ctypes.CDLL(lib_path)
    except OSError as e:
        raise OSError(
            f"Ошибка загрузки библиотеки {lib_path}: {e}\n"
            f"Убедитесь, что библиотека скомпилирована для архитектуры x64 и ваша
            версия Python тоже x64."
        )

    # Получаем функцию collision из библиотеки
    try:
        collision_func = CollisionFunc(collision_lib.collision)
    except AttributeError:
```

```

# В некоторых случаях функция может экспортироваться с подчёркиванием
try:
    collision_func = CollisionFunc(collision_lib._collision)
except AttributeError:
    raise AttributeError(
        f"Функция 'collision' не найдена в библиотеке {lib_name}.\n"
        f"Проверьте экспортируемые символы командой:\n"
        f" Windows: dumpbin /EXPORTS {lib_name}\n"
        f" Linux:   objdump -T {lib_name}\n"
        f" macOS:   nm -gU {lib_name}"
    )

return collision_func

# Создаём глобальную переменную при импорте модуля
asm_collision = load_asm_collision()

```

Файл collision.asm

```

; collision.asm
; Проверка столкновения двух прямоугольников (AABB)
; int collision(int x1, int y1, int w1, int h1, int x2, int y2, int w2, int h2);
; Возвращает 1 если есть пересечение, иначе 0

BITS 64

section .text

%ifidn __OUTPUT_FORMAT__, win64
    global collision
%endif

; Резервируем shadow space (32 байта) и выравниваем стек
collision:
    push    rbp
    mov     rbp, rsp
    sub     rsp, 32                ; shadow space для вызовов (если будем
    ; вызывать что-то ещё)

    ; Параметры:
    ; ecx = x1
    ; edx = y1
    ; r8d = w1
    ; r9d = h1
    ; [rbp+48] = x2 (5-й параметр: rbp+16 (старый rbp) + 8 (адрес возврата) + 32
    ; (shadow) + 8)
    ; [rbp+56] = y2 (6-й параметр)
    ; [rbp+64] = w2 (7-й параметр)
    ; [rbp+72] = h2 (8-й параметр)

    ; Проверка X: x1 + w1 <= x2 ?
    mov     eax, ecx
    add     eax, r8d
    cmp     eax, [rbp+48]
    jle     .no_collision

    ; Проверка X: x2 + w2 <= x1 ?
    mov     eax, [rbp+48]
    add     eax, [rbp+64]
    cmp     eax, ecx
    jle     .no_collision

    ; Проверка Y: y1 + h1 <= y2 ?
    mov     eax, edx
    add     eax, r9d
    cmp     eax, [rbp+56]

```

```

        jle      .no_collision

; Проверка Y: y2 + h2 <= y1 ?
mov     eax, [rbp+56]
add     eax, [rbp+72]
cmp     eax, edx
jle     .no_collision

; Столкновение есть
mov     eax, 1
jmp     .exit

.no_collision:
xor     eax, eax

.exit:
add     rsp, 32
pop     rbp
ret

%else

global collision

collision:
push    rbp
mov     rbp, rsp

; Параметры:
; edi = x1
; esi = y1
; edx = w1
; ecx = h1
; r8d = x2
; r9d = y2
; [rbp+16] = w2 (7-й параметр)
; [rbp+24] = h2 (8-й параметр)

; Проверка X: x1 + w1 <= x2 ?
mov     eax, edi
add     eax, edx
cmp     eax, r8d
jle     .no_collision

; Проверка X: x2 + w2 <= x1 ?
mov     eax, r8d
add     eax, [rbp+16]
cmp     eax, edi
jle     .no_collision

; Проверка Y: y1 + h1 <= y2 ?
mov     eax, esi
add     eax, ecx
cmp     eax, r9d
jle     .no_collision

; Проверка Y: y2 + h2 <= y1 ?
mov     eax, r9d
add     eax, [rbp+24]
cmp     eax, esi
jle     .no_collision

; Столкновение есть
mov     eax, 1
jmp     .exit

.no_collision:
xor     eax, eax

```

```
.exit:
    pop    rbp
    ret
%endif
```