**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра систем автоматизированного проектирования**

**КУРСОВОЙ ПРОЕКТ**

**по дисциплине «Проектирование цифровых устройств»**

**Тема: Разработка аркадной игры Dino Run с использованием Arduino и OLED-дисплея**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3364 |  | Тимошенко Д. Е |
| Студент гр. 3364 |  | Мадраимов Х.Х |
| Студент гр. 3364 |  | Юсфи А. |
| Преподаватель |  | Михайлов А.А. |

Санкт-Петербург

2025

**ЗАДАНИЕ**

**НА КУРСОВОЙ ПРОЕКТ**

|  |  |  |
| --- | --- | --- |
| Студент Тимошенко Д. Е.  Студент Мадраимов Х. Х.  Студент Юсфи А. | | |
| Группа 3364 | | |
| Тема проекта: Разработка аркадной игры Dino Run с использованием Arduino и OLED-дисплея. | | |
| Исходные данные:  Разработать аркадную игру «Dino Run» на базе микроконтроллера Arduino. Устройство должно реализовывать простую игру с графическим интерфейсом на OLED-дисплее, отображающим динозавра, облака и препятствия. Игрок управляет динозавром с помощью кнопки и энкодера: одно нажатие инициирует прыжок, а также переключает игровые режимы (запуск / окончание игры). Счёт отображается на 7-сегментном дисплее (TM1637).  Игровой процесс включает:   * динамическое движение облаков и кактусов; * анимацию прыжка динозавра; * обработку столкновений; * управление звуком через пьезоизлучатель; * отображение экранов начала и окончания игры; * расчёт и вывод текущего счёта в режиме реального времени.   Программная реализация выполнена на языке C++ в среде Arduino IDE с использованием библиотек U8glib.h и TM1637Display.h. Графика реализована с помощью побитовых изображений (bitmaps), хранящихся в памяти PROGMEM. Для удобного взаимодействия с пользователем применяются графические экраны и звуковая индикация.  Для моделирования корпуса устройства возможно использование Autodesk Fusion 360 или аналогичных САПР. Печатная плата при необходимости может быть разработана в EasyEDA или другом редакторе PCB. | | |
| Содержание пояснительной записки:  Содержание, введение, техническое задание, прототип устройства, виртуальная модель устройства, физическая модель устройства, код программы, печатная плата, трехмерная модель корпуса, заключение, список использованных источников, приложение 1 – исходный код программы. | | |
| Предполагаемый объем пояснительной записки:  Не менее 25 страниц. | | |
| Дата выдачи задания: 06.02.2025 | | |
| Дата сдачи реферата: 22.05.2025 | | |
| Дата защиты реферата: 22.05.2025 | | |
| Студент |  | Тимошенко Д. Е. |
| Студент |  | Мадраимов Х. Х. |
| Студент |  | Юсфи А. |
| Преподаватель |  | Михайлов А.А. |

**АННОТАЦИЯ**

Курсовой проект посвящён разработке аркадной игры *Dino Run* на платформе Arduino. Игра с графическим интерфейсом, счётом и звуковыми эффектами управляется с помощью энкодера и отображается на OLED-дисплее. Реализованы анимация, обработка столкновений и смена игровых состояний. Проект демонстрирует применение Arduino в создании встроенных интерактивных систем с ограниченными ресурсами. Разработка способствует развитию навыков в области микроконтроллерного программирования и взаимодействия с периферийными устройствами.

**SUMMARY**

The course project focuses on developing the arcade game *Dino Run* using the Arduino platform. The game features a graphical interface, scoring, and sound effects, controlled via a rotary encoder and displayed on an OLED screen. It includes animation, collision detection, and game state transitions. The project demonstrates the use of Arduino for building interactive embedded systems under limited resources. This development enhances skills in microcontroller programming and peripheral device integration.

**СОДЕРЖАНИЕ**

|  | Введение | 6 |
| --- | --- | --- |
| 1. | Техническое задание | 7 |
| 2. | Прототип устройства | 8 |
| 2.1. | Виртуальная модель устройства | 8 |
| 2.2. | Физическая модель устройства | 10 |
| 3. | Код программы | 12 |
| 4. | Печатная плата | 21 |
| 5. | Трехмерная модель корпуса | 24 |
|  | Заключение | 33 |
|  | Список использованных источников | 35 |
|  | Приложение 1. Исходный код программы | 36 |

**ВВЕДЕНИЕ**

Целью данного проекта является получение и закрепление практических навыков разработки встроенных систем с использованием микроконтроллера Arduino. В рамках работы было создано устройство — компактная игровая консоль, реализующая аркадную игру *Dino Run*, вдохновлённую классической Chrome Dino. Устройство предназначено для демонстрации возможностей Arduino в управлении графическим интерфейсом, взаимодействии с периферийными компонентами и реализации анимации в режиме реального времени.

Формулировка задания:

1. Разработать аппаратную часть устройства, включающую энкодер с кнопкой, OLED-дисплей, 7-сегментный индикатор, звуковой модуль и микроконтроллер Arduino;
2. Реализовать программную часть игры с обработкой игровых состояний, логикой анимации, подсчётом очков и обработкой столкновений;
3. Обеспечить графический вывод на дисплей и управление с использованием минимального пользовательского ввода.

Данный проект способствует освоению принципов работы с микроконтроллерами, программированию интерактивных интерфейсов и обработке событий в реальном времени. Реализованное игровое устройство демонстрирует, как на базе недорогого контроллера можно создавать полноценные системы с визуальной, звуковой и логической составляющей.

**1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Заказчик: кафедра САПР.

Исполнители: Тимошенко Д. Е., Мадраимов Х.Х., Юсфи А.

Цель: Закрепление полученных знаний по курсу «Проектирование цифровых устройств» посредством разработки собственного встроенного устройства — аркадной игры *Dino Run* на базе микроконтроллера Arduino.

Комплектующие:

Проект предполагает использование следующих электронных компонентов:

* микроконтроллерная плата Arduino Uno;
* поворотный энкодер EC11 с кнопкой;
* OLED-дисплей 128×64;
* четырёхразрядный 7-сегментный индикатор (TM1637);
* пьезоизлучатель (buzzer) для звуковых эффектов.

Функциональные требования:

Устройство должно реализовывать:

* вывод игрового процесса на OLED-дисплей;
* подсчёт и отображение текущего счёта на семисегментном индикаторе;
* управление прыжком игрового персонажа с помощью кнопки энкодера;
* смену игровых состояний (старт, игра, завершение) при взаимодействии с пользователем;
* воспроизведение звуковых сигналов при игровых событиях (прыжок, столкновение, переход по уровням).

Нефункциональные требования:

* Надёжность: устойчивость к дребезгу контактов и случайным срабатываниям (в том числе с применением программного фильтра);
* Энергопитание: питание устройства от USB (5 В);
* Эргономика: компактность схемы и удобство взаимодействия с элементами управления;
* Безопасность: защита компонентов от возможных перегрузок и статического электричества.

Программное обеспечение:

* язык программирования: C++;
* среда разработки: Arduino IDE;
* использование библиотек: U8glib (для работы с OLED-дисплеем) и TM1637Display (для 7-сегментного индикатора);
* реализация звуковой индикации с помощью стандартной функции tone().

**2. ПРОТОТИП УСТРОЙСТВА**

**2.1. Виртуальная модель устройства**

На данном этапе была разработана виртуальная модель устройства *Dino Run* на платформе Arduino в онлайн-среде моделирования Wokwi. Работа с симулятором позволила предварительно протестировать основные элементы схемы, проверить взаимодействие компонентов и отладить логику работы программы без риска повреждения физических модулей.

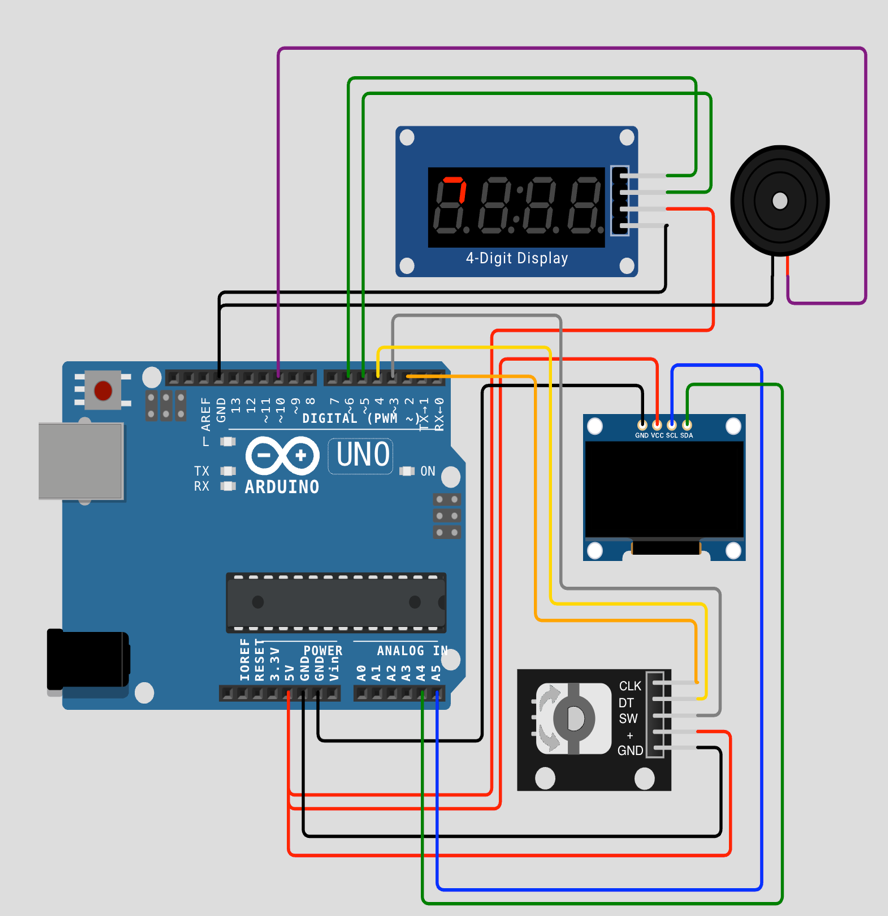
В ходе моделирования была выбрана плата Arduino Uno, поскольку она полноценно поддерживается в Wokwi и обеспечивает корректную работу с основными используемыми компонентами: OLED-дисплеем (128×64, интерфейс I²C), 4-разрядным 7-сегментным индикатором TM1637, поворотным энкодером с кнопкой, а также пьезоизлучателем. Виртуальная среда позволила проверить корректность отображения графики на дисплее, работу звуковых эффектов и функционирование управления персонажем через энкодер.

Создание модели в симуляторе позволило уточнить схему подключения, убедиться в корректности выбранных библиотек и логике анимации, а также минимизировать возможные ошибки до перехода к сборке физической установки.

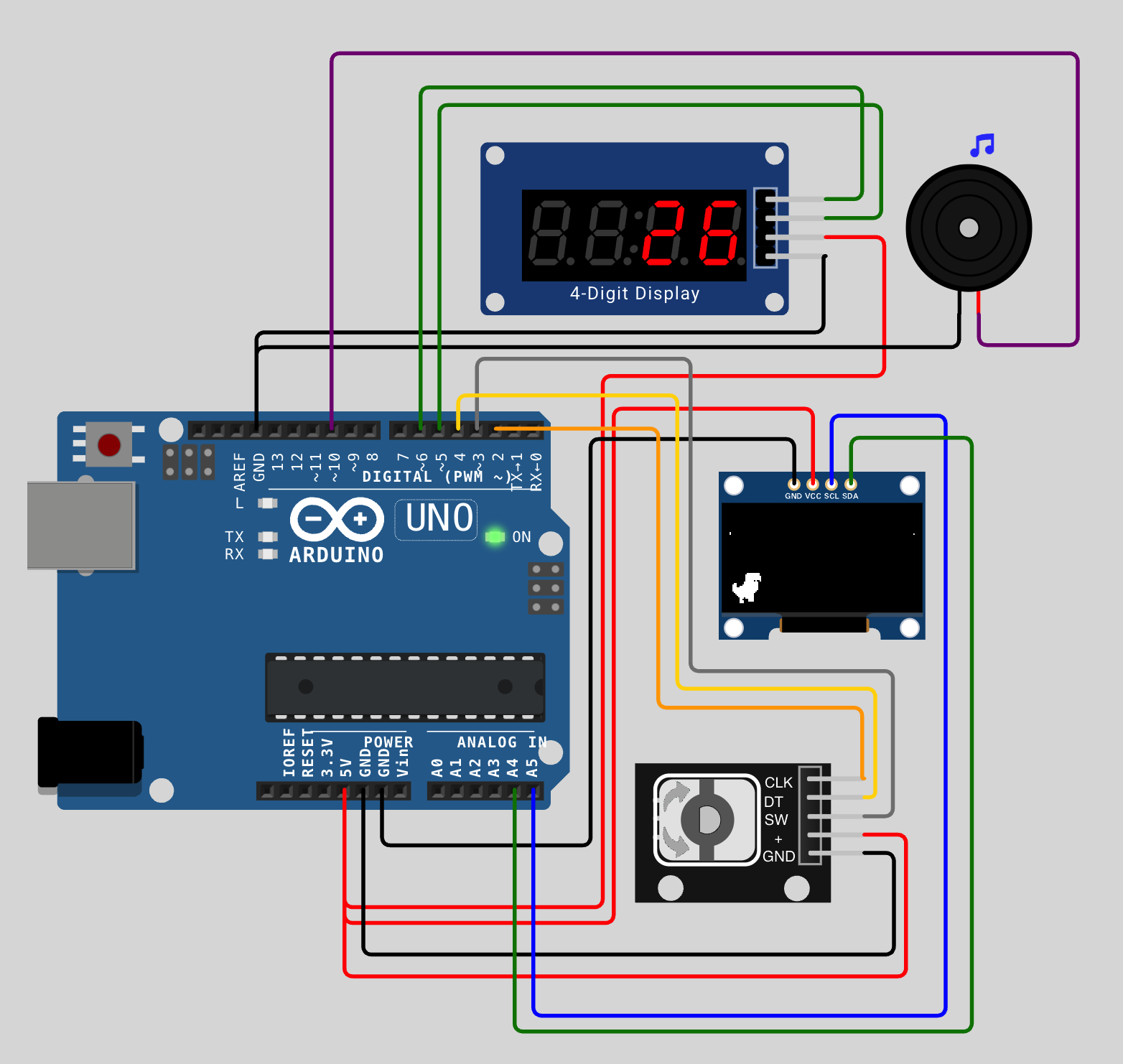
В состав виртуальной модели устройства вошли следующие компоненты:

* Pовортный энкодер EC11 с кнопкой, используемый для управления игровым процессом (прыжок и запуск игры);
* OLED-дисплей 128×64 пикселя, выполняющий функцию основного графического интерфейса, отображающего анимацию, игровые состояния и финальный экран;
* Семисегментный индикатор TM1637 (4 разряда), предназначенный для отображения текущего игрового счёта;
* Пьезоизлучатель, обеспечивающий звуковую индикацию игровых событий (прыжок, столкновение, набор очков).

Для питания и программирования используется USB-порт Arduino, который обеспечивает как передачу данных, так и питание всех компонентов схемы. На рисунке 1 представлена принципиальная схема виртуальной модели устройства, реализованной в среде Wokwi.

  
Рисунок 1 – Схема в Wokwi

В среде Wokwi сборка схемы осуществлялась посредством перетаскивания компонентов на рабочее поле и соединения их проводами в соответствии с логикой подключения. После завершения сборки были разработаны и протестированы несколько версий программного кода для поэтапной проверки работоспособности отдельных элементов устройства: обработка нажатия кнопки энкодера, визуализация графических элементов на OLED-дисплее, вывод счёта на 7-сегментный индикатор, воспроизведение звуковых сигналов с помощью пьезоизлучателя. На рисунке 2 представлен пример работы виртуальной модели в среде Wokwi, демонстрирующий корректное взаимодействие всех компонентов схемы и правильную реализацию логики игрового процесса.

Рисунок 2 – Пример работы модели

Виртуальная модель успешно продемонстрировала работоспособность концепции аркадной игры *Dino Run* и позволила оценить необходимые аппаратные ресурсы и программные зависимости. Моделирование в Wokwi обеспечило безопасную и наглядную среду для отладки схемы и логики игры, выявления потенциальных ошибок и уточнения параметров управления. Таким образом, виртуальная модель стала прочной основой для дальнейшей реализации проекта и позволила уверенно перейти к следующему этапу — созданию физического прототипа устройства.

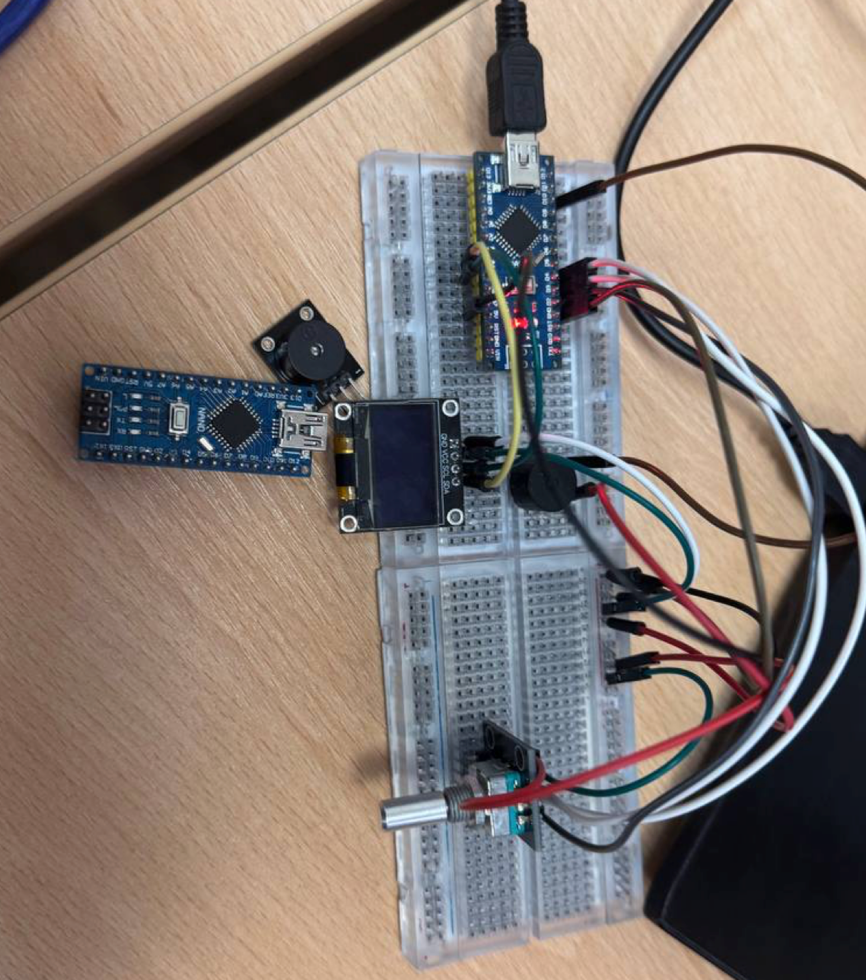
**2.2. Физическая модель устройства**

После успешного тестирования виртуальной модели был разработан и собран физический прототип Macro клавиатуры. Физическая реализация позволила оценить эргономику устройства, тактильные ощущения от нажатия клавиш и общее удобство использования.

Для сборки физического прототипа были использованы следующие компоненты:

* Arduino Uno в качестве основного микроконтроллера;
* Поворотный энкодер EC11 с кнопкой для управления игровым процессом;
* OLED-дисплей SSD1306 с разрешением 128×64 пикселя для вывода графического интерфейса;
* Семисегментный индикатор TM1637 (4-разрядный) для отображения текущего счёта;
* Пьезоизлучатель для воспроизведения звуковых сигналов;
* Макетная плата, соединительные провода и резисторы — для сборки схемы и подключения компонентов.

Сборка прототипа проводилась поэтапно, чтобы ничего не упустить:

1. На макетной плате была собрана основная схема подключения компонентов согласно разработанной виртуальной модели, с незначительными корректировками, учитывающими особенности реальных компонентов;
2. OLED-дисплей был установлен в нижней части устройства для удобного считывания информации и навигации по меню.

Итог сборки физического прототипа представлен на рисунке 3.

Рисунок 3 – Физическая модель устройства

После сборки устройства была выполнена первичная прошивка микроконтроллера с использованием тестового кода, направленного на поэтапную проверку работы всех подключённых компонентов. В процессе тестирования были выявлены и устранены следующие недостатки:

1. Обнаружена нестабильность в работе энкодера, вызванная дребезгом контактов. Для устранения проблемы было реализовано программное подавление дребезга;
2. Принято решение на этапе разработки печатной платы дополнительно реализовать аппаратное подавление дребезга с помощью RC-фильтра;
3. Выявлено ограничение объёма памяти у микроконтроллера Arduino Pro Micro, что не позволяет использовать полноценную анимацию с высокой частотой обновления. В результате было решено сосредоточиться на текстовом отображении текущего режима и функций кнопок.

Созданный физический прототип подтвердил жизнеспособность технической концепции и позволил получить ценный опыт для последующего этапа — проектирования печатной платы и финальной доработки устройства.

Основные преимущества разработанного прототипа:

* Компактные габариты, обеспечивающие удобство эксплуатации;
* Тактильная обратная связь при взаимодействии с элементами управления;
* Наглядное отображение состояния и игровой информации на OLED-дисплее.

**3. КОД ПРОГРАММЫ**

Разработка программного обеспечения для аркадной игры *Dino Run* осуществлялась в среде Arduino IDE с использованием языка программирования C++. Программа структурирована по модульному принципу, что обеспечивает удобство отладки, повторного использования кода и возможность дальнейшего расширения функциональности, включая добавление новых игровых механик, улучшение анимации и звукового сопровождения.

1. **Подключение библиотек и определение констант**

Программа начинается с подключения необходимых библиотек:

// Библиотека для OLED-дисплея SH1106 (совместима с SSD1306)

#include <U8glib.h>

// Библиотека для 4-разрядного 7-сегментного индикатора TM1637

#include "TM1637Display.h"

Далее объявляются основные пины устройства и константы:

// Пины энкодера EC11

#define PIN\_SW   3   // Кнопка энкодера

#define PIN\_DT   4   // Вывод DT

#define PIN\_CLK  2   // Вывод CLK

// Пины 7-сегментного индикатора TM1637

#define PIN\_DCLK 6   // CLK индикатора

#define PIN\_DIO  5   // DIO индикатора

// Пин пьезоизлучателя

#define PIN\_BUZZER 10

// Константы звуковых эффектов (в Гц)

#define TONE\_JUMP   700

#define TONE\_HIT    125

#define TONE\_LEVEL 1000

// Константа подавления дребезга кнопки (мс)

#define DEBOUNCE\_DELAY 180

// Константы состояний игры

#define GAME\_START   0

#define GAME\_END     1

#define GAME\_PLAYING 2

**U8glib.h** используется для вывода графики на OLED-дисплей (анимация динозавра, облаков и кактусов).

**TM1637Display.h** управляет семисегментным индикатором, на котором отображается счёт.

Пины энкодера задают управление прыжком и запуском игры, а пьезоизлучатель воспроизводит звуковые сигналы.

1. **Настройка пинов и переменных**

// Объект 7-сегментного индикатора

TM1637Display scoreDisplay(PIN\_DCLK, PIN\_DIO);

// Массив для гашения всех сегментов (используется при очистке)

const uint8\_t SEG\_BLANK[4] = {0x00, 0x00, 0x00, 0x00};

// Объект OLED-дисплея (I²C, адрес 0x3C по умолчанию)

U8GLIB\_SH1106\_128X64 oled(U8G\_I2C\_OPT\_NONE);

// Служебные переменные игры

volatile int gameStatus = GAME\_START;

int dinoY = 0; // Текущее смещение динозавра по вертикали

volatile int jump = 0; // Флаг прыжка (0 – нет, 1/2 – фаза прыжка)

int score = 0; // Текущий счёт

int speed = 8; // Скорость движения препятствий

unsigned long tStart = 0; // Время старта партии

Переменные gameStatus, jump, score и speed обеспечивают логику игрового процесса.

Пины энкодера и индикатора объявлены через #define, что упрощает изменение конфигурации при необходимости.

Константа DEBOUNCE\_DELAY (180 мс) задаёт время программного подавления дребезга кнопки энкодера.

1. **Инициализация системы (функция setup)**

void setup() {

// Инициализация 7-сегментного индикатора TM1637

OurDisplay.setBrightness(7); // максимальная яркость

OurDisplay.clear(); // очистка при старте

// Сброс игровых переменных и вывод счёта «0»

resetGame();

ShowScore();

// Настройка пинов энкодера и кнопки

pinMode(PIN\_CLK, INPUT); // вывод CLK энкодера

pinMode(PIN\_DT, INPUT); // вывод DT энкодера

pinMode(PIN\_SW, INPUT\_PULLUP); // кнопка энкодера с подтяжкой

// Настройка пина пьезоизлучателя

pinMode(PIN\_BUZZER, OUTPUT);

// Прерывание по нажатию кнопки (запуск/прыжок/сброс)

attachInterrupt(digitalPinToInterrupt(PIN\_SW),

StartStopGame, FALLING);

}

Функция setup() выполняется один раз при подаче питания.

Сначала настраивается семисегментный индикатор: задаётся яркость, выводится нулевой счёт. Затем инициализируются пины энкодера и кнопки; для PIN\_SW используется внутренний подтягивающий резистор (INPUT\_PULLUP). Пьезоизлучатель готов к воспроизведению звуков, а прерывание на кнопке сразу связывает управление прыжком и запуском игры.

1. **Основной цикл программы**

void loop() {

    // Обновление счёта на 7-сегментнике

    ShowScore();

    // Отрисовка текущего кадра на OLED-дисплее

    My\_u8g\_Panel.firstPage();

    do {

        draw();              // вывод графики и текстов

    } while (My\_u8g\_Panel.nextPage());

    // Обновление объектов игры, если идёт игровая сессия

    if (gameStatus == GAME\_PLAYING) {

        moveDino();          // анимация прыжка

        moveCloud();         // движение облака

        moveObstacles();     // сдвиг кактусов и генерация новых

    }

}

Цикл loop() работает непрерывно.

Каждая итерация выполняет три ключевые задачи:

1. Вывод счёта – функция ShowScore() вычисляет очки по времени и отражает их на TM1637.

2. Рендеринг кадра – библиотека U8glib постранично выводит графику (draw()), обеспечивая плавную анимацию на OLED-дисплее.

3. Логика игры – если статус GAME\_PLAYING, обновляются координаты динозавра, облака и препятствий; при этом проверяются столкновения и возрастает сложность по мере роста скорости.

1. **Обработка нажатия кнопки энкодерак.**

// Константа подавления дребезга (мс)

#define DEBOUNCE\_DELAY 180

void StartStopGame() {

static unsigned long lastInterrupt = 0;

// Прерывание игнорируется, если с предыдущего прошло < DEBOUNCE\_DELAY

if (millis() - lastInterrupt < DEBOUNCE\_DELAY) return;

lastInterrupt = millis();

if (gameStatus == GAME\_PLAYING) { // во время игры — прыжок

if (jumping == 0) { // если динозавр на земле

jumping = 1; // запускаем фазу прыжка

tone(PIN\_BUZZER, TONE\_JUMP, 100); // короткий звуковой эффект

}

}

else if (gameStatus == GAME\_START) { // экран приветствия

gameStatus = GAME\_PLAYING; // начинаем игру

}

else { // GAME\_END → возврат к старту

gameStatus = GAME\_START;

}

}

**Алгоритм работы:**

1. Первые 180 мс после прерывания вход игнорируется — это простое программное подавление дребезга.
2. Если игра уже идёт (GAME\_PLAYING), короткое нажатие наносит импульс персонажу вверх (jumping = 1).
3. На экране приветствия (GAME\_START) то же нажатие запускает партию, а на экране «Game Over» переводит игру в исходное состояние.

.

1. **StartStopGame() — обработка нажатия кнопки энкодера**

* Исполняется каждый цикл loop(); активна только при GAME\_PLAYING.
* Вычисляет очки по формуле: ((millis() - tStart) \* speed) / 1000.
* Выводит число на TM1637; каждые 100 очков проигрывает звуковую пару тонов.

void ShowScore() {

    if (gameStatus != GAME\_PLAYING) return;

    curTime = millis();

    score   = (curTime - tStart) \* speed / 1000;

    scoreDisplay.showNumberDecEx(score);

    if (score / 100 > lastBeep) {             // каждые 100 очков

        tone(PIN\_BUZZER, 1000, 100);

        delay(150);

        tone(PIN\_BUZZER, 1250, 100);

        lastBeep = score / 100;

    }

}

Позволяет игроку отслеживать результат на внешнем индикаторе независимо от графики OLED.

1. **moveDino() — анимация прыжка**

* При jumping == 0 циклически переключает два кадра шага динозавра.
* При начале прыжка (jumping == 1) поднимает спрайт по 8 пикс./кадр.
* После достижения вершины (dinoMove > 32) переводит в фазу спуска (jumping == 2) до возврата на землю.
* После каждого сдвига вызывает checkCollision() для детекции удара.

void moveDino() {

    if (jumping == 0) dinoMove = (dinoMove + 1) % 3;      // шаги

    else {

        if (jumping == 1) {                               // подъём

            dinoMove += 8;

            if (dinoMove > 32) jumping = 2;

        } else {                                          // спуск

            dinoMove -= 8;

            if (dinoMove < 8) { jumping = 0; dinoMove = 0; }

        }

    }

    checkCollision();

}

Функция полностью описывает кинематику прыжка без использования float-арифметики.

.

1. **moveCloud() — движение облака**

* Сдвигает координату cloudx на 1 пиксель влево каждый кадр.
* При выходе облака за левый край (cloudx < −38) переносит его на 128.

void moveCloud() {

    cloudx--;

    if (cloudx < -38) cloudx = 128;

}

Обеспечивает бесконечную «прокрутку» фона без дополнительных массивов.

1. **moveObstacles() — управление кактусами**

* Смещает ближайший кактус на speed пикс./кадр.
* Когда координата obx < −20, формирует новый интервал (80–125 px) и случайный тип как­туса 1…6.
* Всегда поддерживает на экране ровно два препятствия.

void moveObstacles() {

    int obx = obstaclex[0] - speed;

    if (obx < -20) {                       // старый ушёл — генерируем новый

        obstaclex[0] = obstaclex[1];

        obstaclex[1] = obstaclex[0] + random(80, 125);

        obstacles[0]  = obstacles[1];

        obstacles[1]  = random(1, 6);

    } else {

        obstaclex[0] = obx;

        obstaclex[1] -= speed;

    }

}

Схема проста и надёжна: всегда есть «первый» и «второй» кактус с гарантированным зазором.

1. **checkCollision() — обнаружение столкновения**

* По ID кактуса определяет ширину obw и высоту obh спрайта.
* Проверяет горизонтальное перекрытие (5 < x < 15) и недопустимую высоту прыжка (dinoMove ≤ obh − 3).
* В случае столкновения устанавливает GAME\_END и проигрывает двойной звук

void checkCollision () {

int obx = obstaclex [0];

int obw, obh;

switch (obstacles[0]) {

case 0: obw = 39; obh = 10; break;

case 1: obw = 10; obh = 20; break;

case 2: obw = 17; obh = 20; break;

case 3: obw = 17; obh = 20; break;

case 4: obw = 6; obh = 12; break;

case 5: obw = 12; obh = 12; break;

case 6: obw = 17; obh = 12; break;

}

if (obx > 15 || obx + obw < 5 || dinoMove > obh - 3) {}

else {

gameStatus = gameEnd;

tone (buzzer, 125, 100);

delay(150);

tone (buzzer, 125, 100);

}

}

void drawObsticles () {

drawShape (obstacles[0], obstaclex[0]);

drawShape (obstacles[1], obstaclex[1]);

}

void u8g\_prepare(void) {

My\_u8g\_Panel.setFont(u8g\_font\_6x10);

My\_u8g\_Panel.setFontRefHeightExtendedText();

My\_u8g\_Panel.setDefaultForegroundColor();

My\_u8g\_Panel.setFontPosTop();

}

Алгоритм использует прямоугольную маску: быстро и достаточно точно для монохромной аркады.

1. **draw() — формирование кадра**

* Вызывает u8g\_prepare() для настройки шрифта и цвета.
* В зависимости от gameStatus выводит:

GAME\_PLAYING → игровой кадр;

GAME\_START  → заставка «Welcome to Dino»;

GAME\_END    → баннер «Game Over».

* Игровой кадр строится через drawDino(), drawShape() (облако) и drawObsticles().

void draw() {

u8g\_prepare();

if (gameStatus == GAME\_PLAYING) {

drawDino();

drawShape(0, cloudx);

drawObsticles();

} else if (gameStatus == GAME\_START) {

My\_u8g\_Panel.drawStr(0, 10, "Welcome to");

My\_u8g\_Panel.drawStr(10, 30, "Dino!!");

My\_u8g\_Panel.drawStr(0, 50, "Push to begin");

} else {

My\_u8g\_Panel.drawXBMP(14, 12, 100, 15, gameOver);

drawDino();

drawShape(0, cloudx);

drawObsticles();

}

}

Все вызовы U8glib сведены в одно место, что обеспечивает единый контроль за выводом.

1. **drawDino() — выбор спрайта динозавра**

* При GAME\_END выводит позу «Blah».
* На земле циклически переключает кадры шага (dinoLeft, dinoRight).
* В прыжке (любая Y-позиция ≠ 0) использует спрайт dinoJump.

void drawDino() {

if (gameStatus == GAME\_END) {

My\_u8g\_Panel.drawXBMP(0, 43 - dinoMove, 20, 21, dinoBlah);

return;

}

switch (dinoMove) { // -1: Blah, 0-2: шаги, >2: прыжок

case 0: My\_u8g\_Panel.drawXBMP(0, 43, 20, 21, dinoJump); break;

case 1: My\_u8g\_Panel.drawXBMP(0, 43, 20, 21, dinoLeft); break;

case 2: My\_u8g\_Panel.drawXBMP(0, 43, 20, 21, dinoRight); break;

default: My\_u8g\_Panel.drawXBMP(0, 43 - dinoMove, 20, 21, dinoJump); break;

}

}

Отдельные спрайты хранятся в PROGMEM, что экономит ОЗУ микроконтроллера

1. **drawShape() — универсальный вывод спрайтов**

* Получает тип объекта shape и X-координату.
* Выводит соответствующий XBM-спрайт (облако или один из шести кактусов).

void drawShape (int shape, int x) {

switch (shape) {

case 0: My\_u8g\_Panel.drawXBMP( x, 5, 39, 12, cloud); break;

case 1: My\_u8g\_Panel.drawXBMP( x, 44, 10, 20, oneCactus); break;

case 2: My\_u8g\_Panel.drawXBMP( x, 44, 20, 20, twoCactus); break;

case 3: My\_u8g\_Panel.drawXBMP( x, 44, 20, 20, threeCactus); break;

case 4: My\_u8g\_Panel.drawXBMP( x, 52, 6, 12, oneCactusSmall); break;

case 5: My\_u8g\_Panel.drawXBMP( x, 52, 12, 12, twoCactusSmall); break;

case 6: My\_u8g\_Panel.drawXBMP( x, 52, 17, 12, threeCactusSmall); break;

}

}

Используется для облака (shape = 0) и в drawObsticles() для кактусов

1. **drawObsticles() — вывод пары кактусов**

void drawObsticles() {

    drawShape(obstacles[0], obstaclex[0]);

    drawShape(obstacles[1], obstaclex[1]);

}

Минимальный «проксивызов» — держит логику отрисовки препятствий отдельно от расчёта позиций.

1. **u8g\_prepare() — подготовка OLED-дисплея**

* Выбирает шрифт u8g\_font\_6x10.
* Включает режим расширенной высоты символов.
* Устанавливает белый цвет рисования и позицию текста «по верхнему краю».

void u8g\_prepare() {

    My\_u8g\_Panel.setFont(u8g\_font\_6x10);

    My\_u8g\_Panel.setFontRefHeightExtendedText();

    My\_u8g\_Panel.setDefaultForegroundColor();

    My\_u8g\_Panel.setFontPosTop();

}

Гарантирует единообразие текста на всех экранах (заставка, Game Over, счёт и т.д.).

**Особенности реализации**

1. Неблокирующее выполнение: Все задержки реализованы через сравнение времени с millis(), что позволяет плавно обновлять графику, счёт и реагировать на ввод без пауз.
2. Программное подавление дребезга: Контакты кнопок обрабатываются с учётом временных фильтров, что предотвращает ложные срабатывания при прыжке или запуске игры.
3. Модульная структура кода: Логика разбита на отдельные функции: отрисовка, физика движения, генерация препятствий, обработка коллизий и счёта — это упрощает отладку и добавление новых функций.
4. Оптимизация производительности: Отрисовка и вычисления обновляются с минимальной нагрузкой на микроконтроллер, благодаря простым алгоритмам и предопределённым спрайтам в PROGMEM.

Разработанная прошивка реализует полный цикл аркадной игры на базе Arduino, включая управление персонажем, динамическое появление препятствий, систему подсчёта очков и графическую обратную связь на OLED-дисплее.

**4. ПЕЧАТНАЯ ПЛАТА**

На основе успешно протестированного прототипа и отлаженной схемы устройства была разработана печатная плата для проекта *Dino Run*. Проектирование велось в среде **EasyEDA**, которая предоставляет удобные инструменты для создания схем, трассировки дорожек и подготовки гербер-файлов для последующего производства платы.

Разработка печатной платы проходила в несколько этапов:

1. **Создание принципиальной схемы:** На первом этапе была составлена подробная принципиальная схема, включающая все элементы устройства: микроконтроллер **Arduino Nano**, поворотный **энкодер с кнопкой**, **7-сегментный дисплей TM1637**, **OLED-дисплей по шине I²C**, а также **пьезоизлучатель** для звуковой индикации. Составленная схема (рис. 4) легла в основу дальнейшей разводки печатной платы и обеспечила корректное подключение всех компонентов, включая фильтрацию сигналов от энкодера и надёжное питание периферии.

Изображение выглядит как текст, диаграмма, линия, План

Автоматически созданное описание

Рисунок 4 – Принципиальная схема устройства

1. **Выбор и размещение компонентов:**

Для проектирования печатной платы были выбраны следующие основные компоненты и их модели:

* Микроконтроллер: Arduino Nano на базе ATmega328P — обеспечивает управление всеми элементами системы.
* Энкодер: PEC11R-4015F-S0024 — механический поворотный энкодер с кнопкой, используется для управления игрой.
* Дисплей: 4-разрядный 7-сегментный индикатор TM1637, подключаемый по двухпроводному интерфейсу.
* OLED-дисплей: HS154L03W2C01 — графический дисплей с интерфейсом I²C (SCL, SDA).
* Пьезоизлучатель: BZ1 — используется для подачи звуковых сигналов (прыжок, проигрыш и т. д.).
* Фильтрация сигналов энкодера: два конденсатора по 100 нФ и резисторы 10 кОм для подавления дребезга контактов.

1. **Трассировка печатной платы:** Разводка печатной платы осуществлялась в среде **EasyEDA** в один слой (односторонняя плата). При проектировании учитывались стандартные требования к печатным проводникам и компоновке элементов:

* Минимальная ширина сигнальных дорожек — 0,5 мм
* Минимальная ширина силовых дорожек — 0,5 мм
* Минимальный зазор между дорожками — 0,5 мм

1. **Проверка и оптимизация:** после завершения трассировки печатной платы (см. рис. 5) была проведена серия проверок и улучшений, направленных на обеспечение надежной работы устройства:

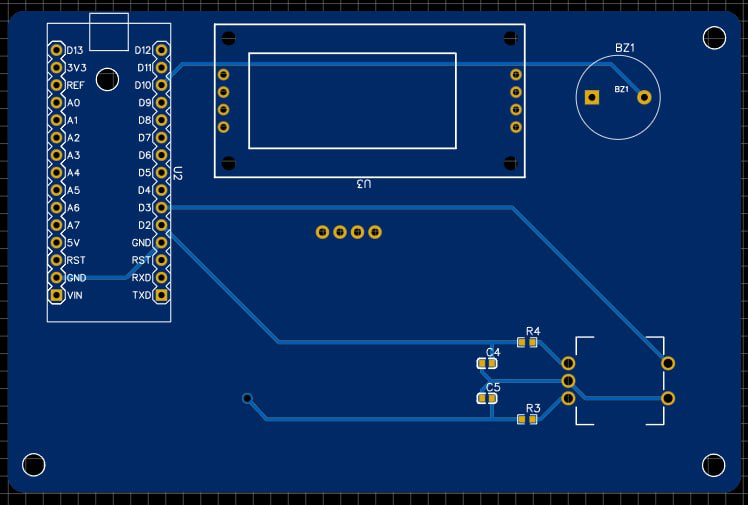
* Проверка DRC (Design Rule Check): выявлены и устранены возможные нарушения правил проектирования, включая ширину дорожек, зазоры и перекрытия
* Анализ целостности сигналов: проведён визуальный контроль ключевых соединений, таких как линии питания, дисплей TM1637 и интерфейс энкодера.
* Оптимизация размещения компонентов: изменено расположение некоторых элементов для сокращения длины соединений и упрощения пайки при сборке.
  + 

Рисунок 5 – Плата с трассировкой

1. **3D-визуализация:** Для проверки физического размещения всех компонентов и оценки их взаимодействия с возможным корпусом устройства была создана 3D-модель печатной платы с установленными элементами в среде EasyEDA (см. рис. 6 и 7). Это позволило убедиться в правильности ориентации разъёмов, достаточном зазоре между деталями и удобстве пайки.

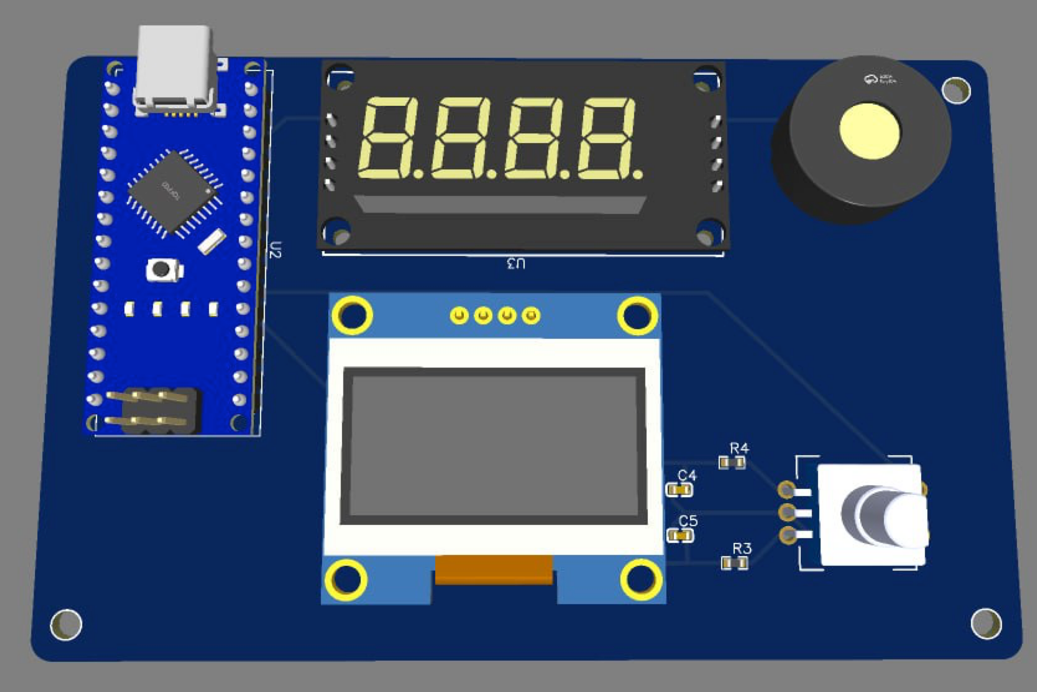


Рисунок 6 – 3D модель платы (верхний слой)

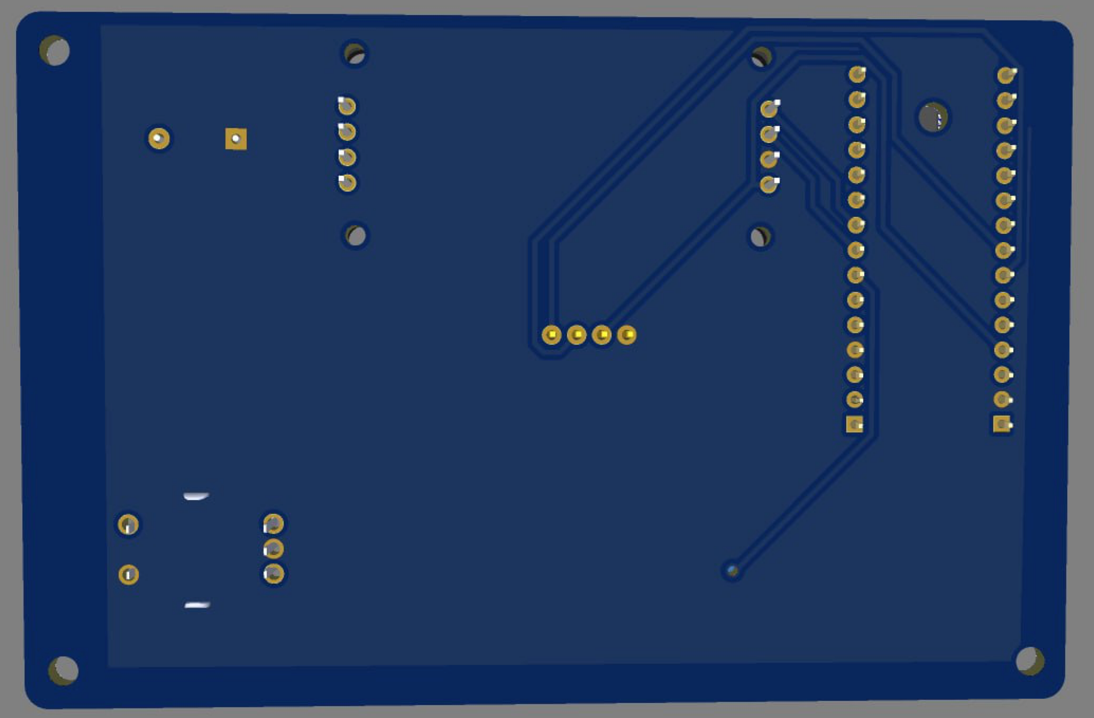


Рисунок 7 – 3D модель платы (нижний слой)

Итоговые размеры печатной платы составили 107×70 мм, что обеспечивает компактность игрового устройства и удобное размещение всех элементов — энкодера, дисплея TM1637, зуммера и Arduino Nano. Плата содержит 4 монтажных отверстия диаметром 3 мм для крепления внутри корпуса. Разработанная конструкция полностью соответствует требованиям проекта Dino Run и готова к изготовлению с использованием стандартных технологий производства печатных плат.

**5. ТРЕХМЕРНАЯ МОДЕЛЬ КОРПУСА**

Завершающим этапом разработки игрового устройства Dino Run стало создание трехмерной модели корпуса, обеспечивающего защиту всех электронных компонентов, удобство управления и визуальную привлекательность. Проектирование корпуса осуществлялось в среде Fusion 360 с учётом габаритов печатной платы, расположения энкодера, дисплея и зуммера. Модель корпуса была разработана таким образом, чтобы обеспечить комфортную игру и лёгкий доступ к управляющим элементам.

**5.1. Концепция и дизайн корпуса**

Первым этапом проектирования корпуса для игрового устройства Dino Run стало формирование общей концепции, соответствующей следующим требованиям:

1. **Эргономичность** —корпус должен обеспечивать удобное управление энкодером, хорошую видимость дисплея и надёжное удержание устройства в руках во время игры.
2. **Защита компонентов** —корпус обязан защищать электронику от механических повреждений, пыли и возможных внешних воздействий.
3. **Визуальная привлекательность** — дизайн корпуса выполнен в простой и современной стилистике, соответствующей духу аркадных игр.
4. **Производственная пригодность** — конструкция учитывает особенности 3D-печати и допускает легкую сборку всех компонентов.

На основе этих требований был создан эскиз корпуса, представленный на рисунке 8.

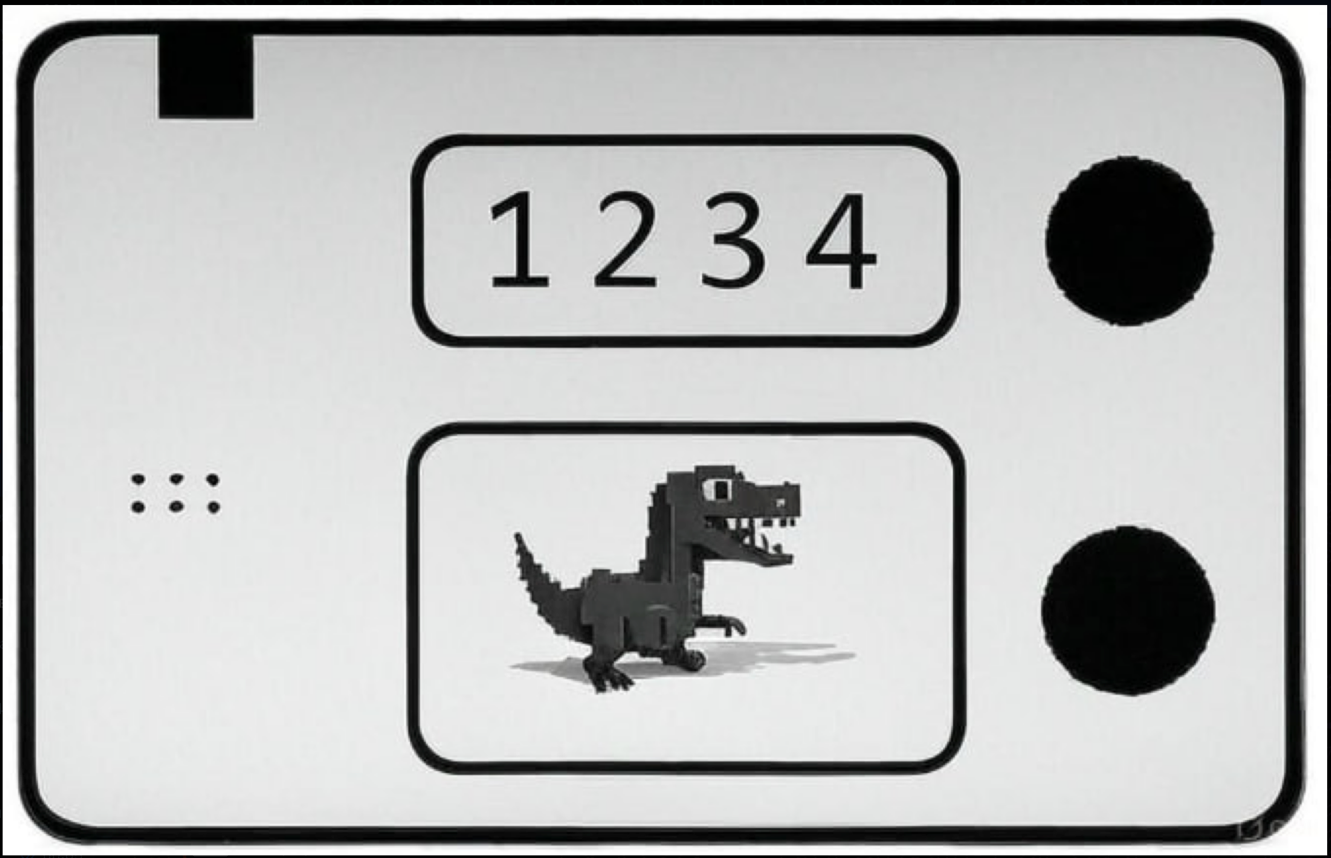


Рисунок 8 – Эскиз корпуса для устройства

**5.2. Моделирование нижней части корпуса**

Процесс моделирования нижней части корпуса для устройства Dino Run начался с формирования основы. Работа включала следующие этапы:

1. Создание эскиза основания на плоскости XY с размерами 112×76 мм, которое затем было выдавлено на высоту 2 мм.
2. На верхней грани основания создан эскиз внутреннего пространства с отступами 3 мм от края для формирования стенок. Этот эскиз был выдавлен вверх на 9 мм.
3. В нижней части добавлены четыре цилиндрических выступа высотой 5 мм с отверстиями диаметром 3 мм для крепления печатной платы.
4. На боковых стенках сделаны выступы для соединения с верхней частью корпуса.

Результат моделирования нижней части корпуса представлен на рисунках 9 и 10.

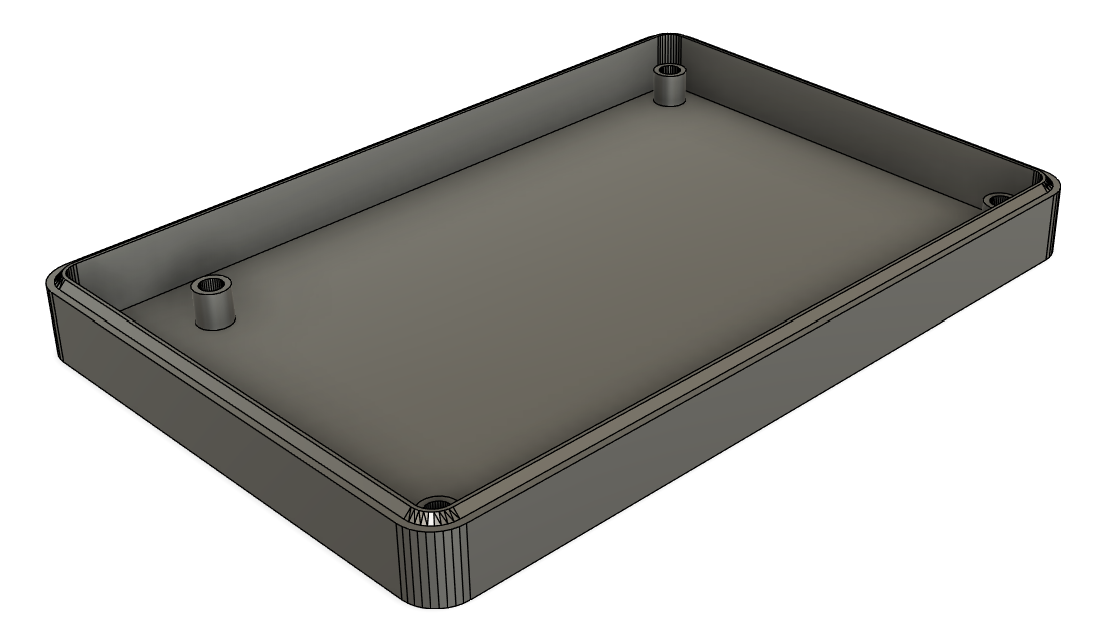


Рисунок 9 – Нижняя часть корпуса (вид сбоку)

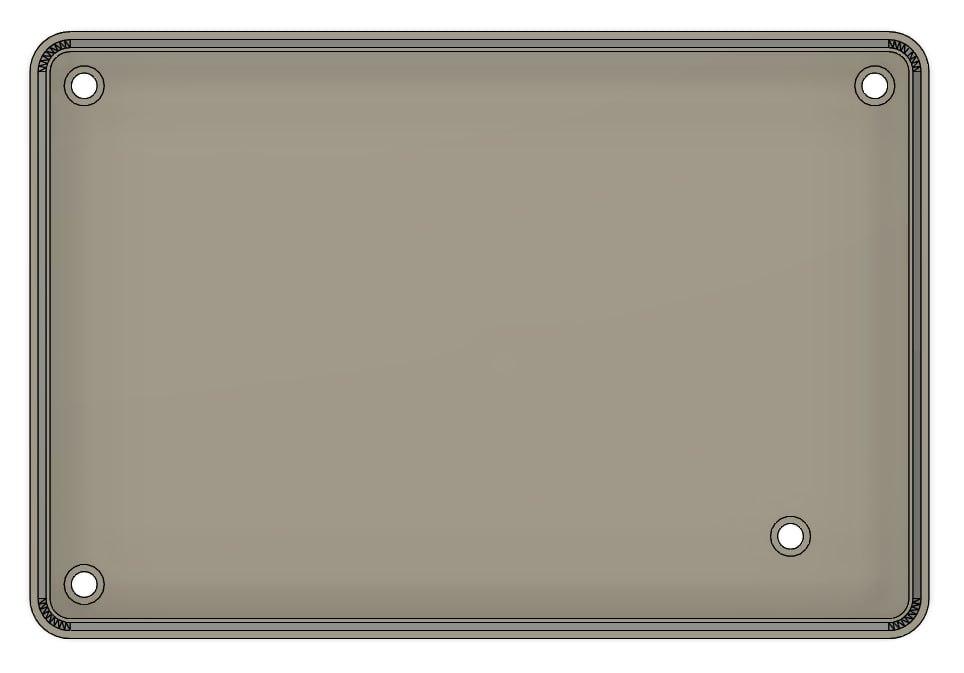


Рисунок 10 – Нижняя часть корпуса (вид сверху)

### 

**5.3. Моделирование верхней части корпуса**

Верхняя часть корпуса устройства Dino Run выполняет роль лицевой панели и разрабатывалась с учётом размещения элементов управления

1. Создан эскиз основания панели с размерами, совпадающими с нижней частью, и выдавлен на высоту 6 мм.
2. На верхней поверхности спроектированы отверстия для трёх механических клавиш (16×16 мм), энкодера EC11 (диаметр 10 мм) и OLED-дисплея (36×20 мм), что обеспечивает удобный доступ к элементам управления.
3. По боковым сторонам добавлены выступы и пазы для надёжного соединения с нижней частью корпуса.

Результат моделирования верхней части корпуса представлен на рисунках 11, 12.

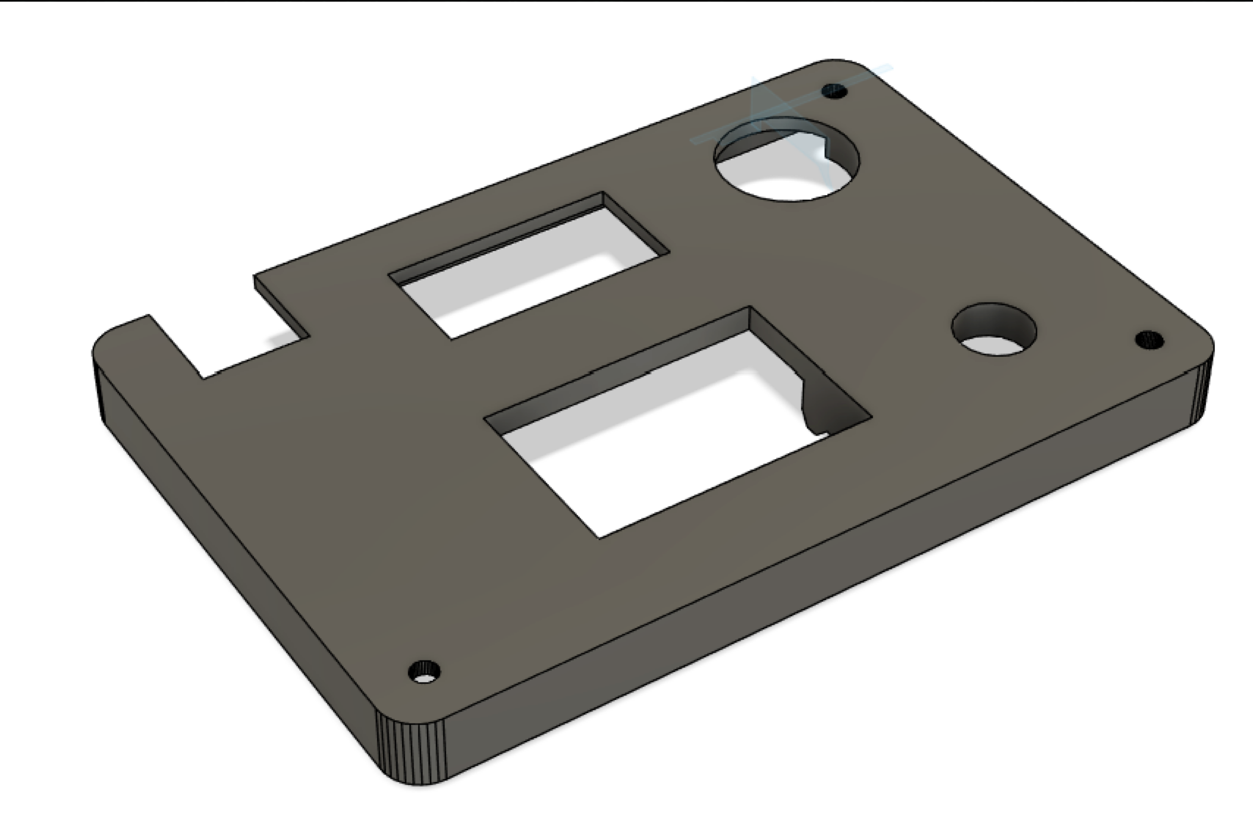


Рисунок 11 – Верхняя часть корпуса (вид сбоку)

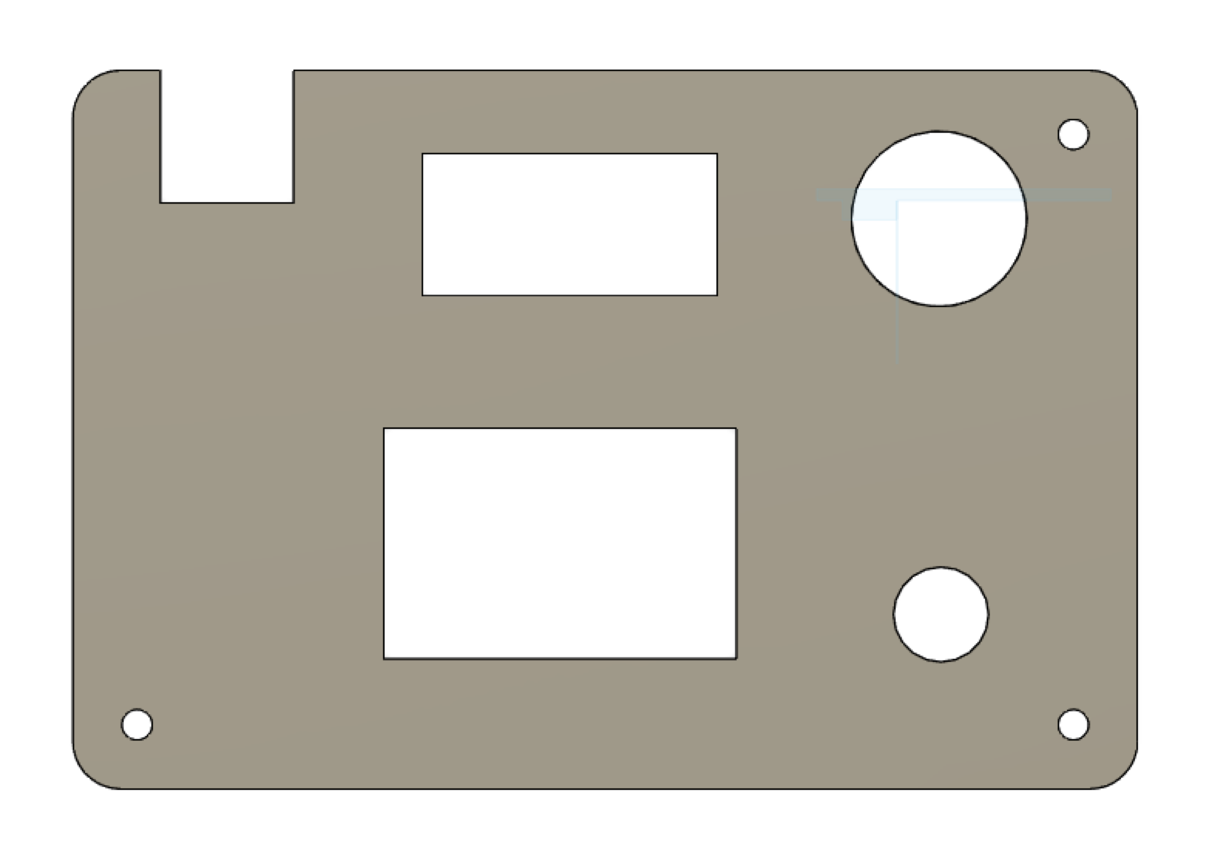


Рисунок 12 – Верхняя часть корпуса (вид сверху)

**5.4. Сборка и анализ модели**

Финальным этапом проектирования корпуса для устройства «Dino Run» стала виртуальная сборка модели и её проверка на соответствие всем техническим требованиям. В процессе были выполнены следующие действия:

1. В новом проекте Fusion 360 были объединены трёхмерные модели верхней и нижней частей корпуса;
2. Детали корпуса были сопряжены по внешним граням и монтажным отверстиям, обеспечивая точную стыковку;
3. Проведена проверка корректности размещения дисплея TM1637, OLED-экрана и энкодера в отведённых местах;
4. При выявлении нестыковок в габаритах компонентов были внесены необходимые правки в эскизы;
5. Для завершённого вида были добавлены декоративный колпачок на энкодер;
6. Визуальная финализация выполнена в виде 3D-рендера, демонстрирующего корпус с установленной платой

Результат сборки модели корпуса представлен на рисунках 13 – 17.

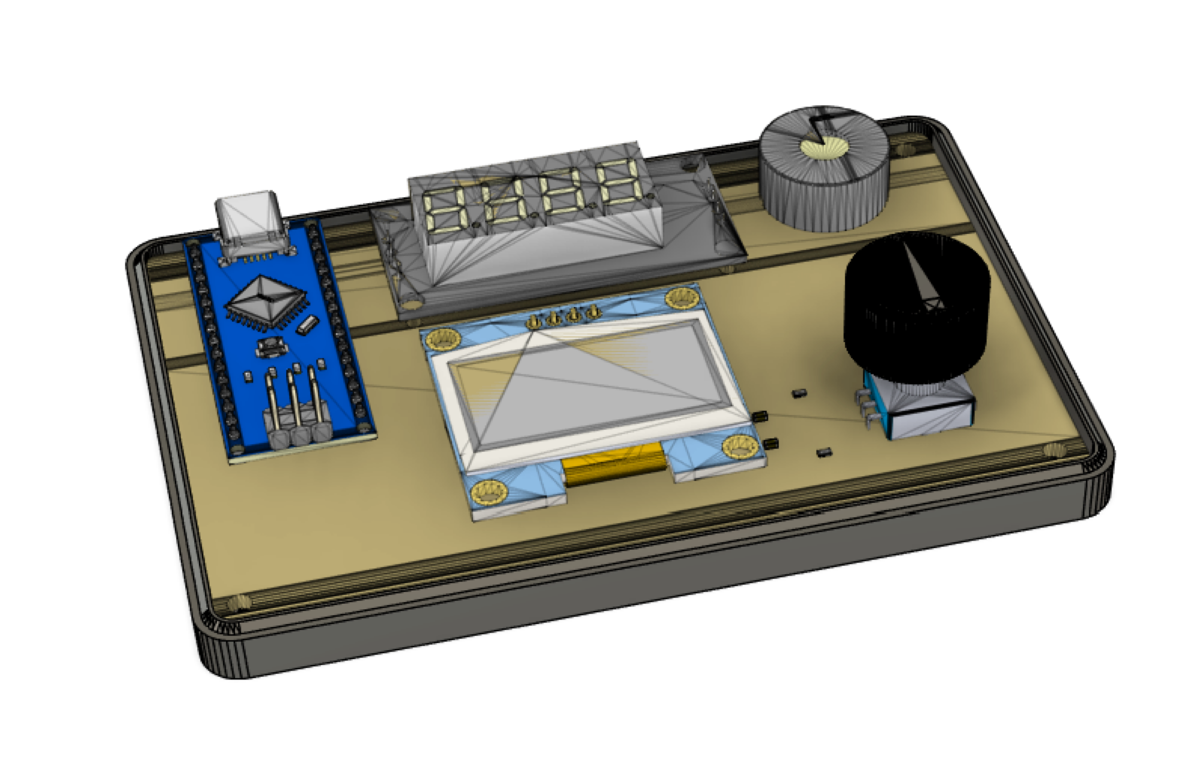


Рисунок 13 – Нижняя часть корпуса с платой внутри (вид сверху)

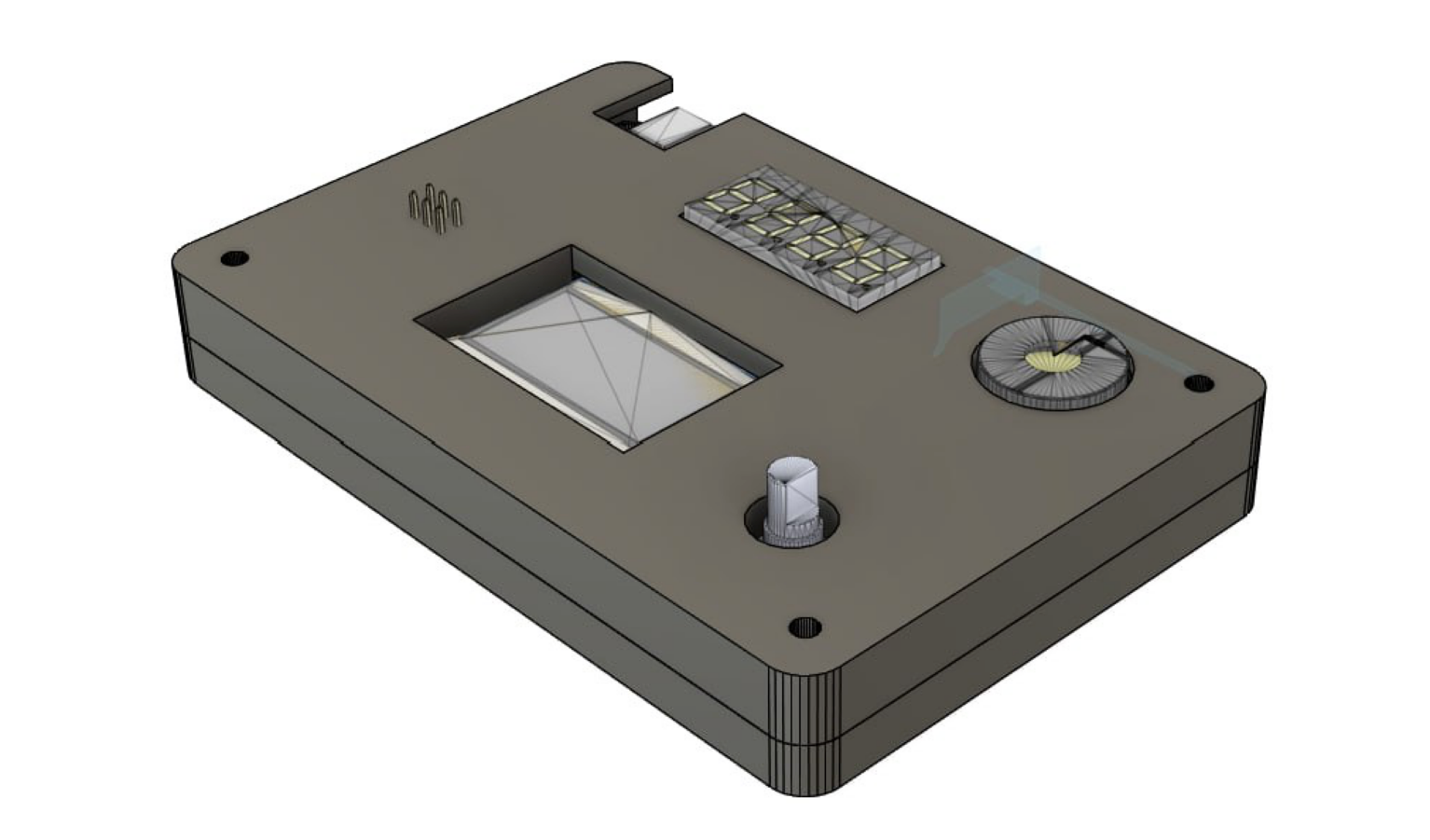


Рисунок 14 – Готовая модель корпуса

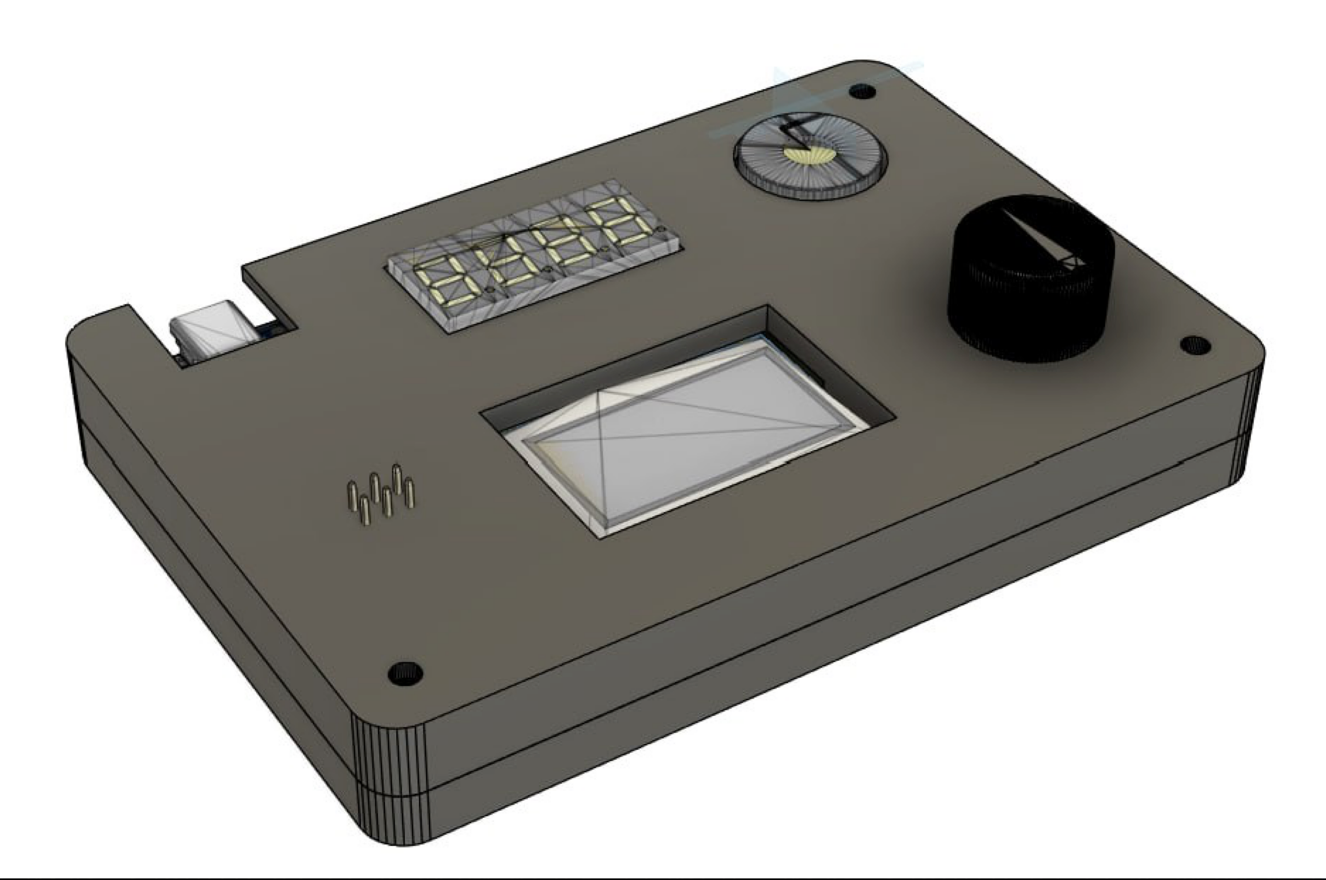


Рисунок 15 – Готовая модель корпуса c колпачками



Рисунок 16 – Рендер готовой модели корпус (вид 1)

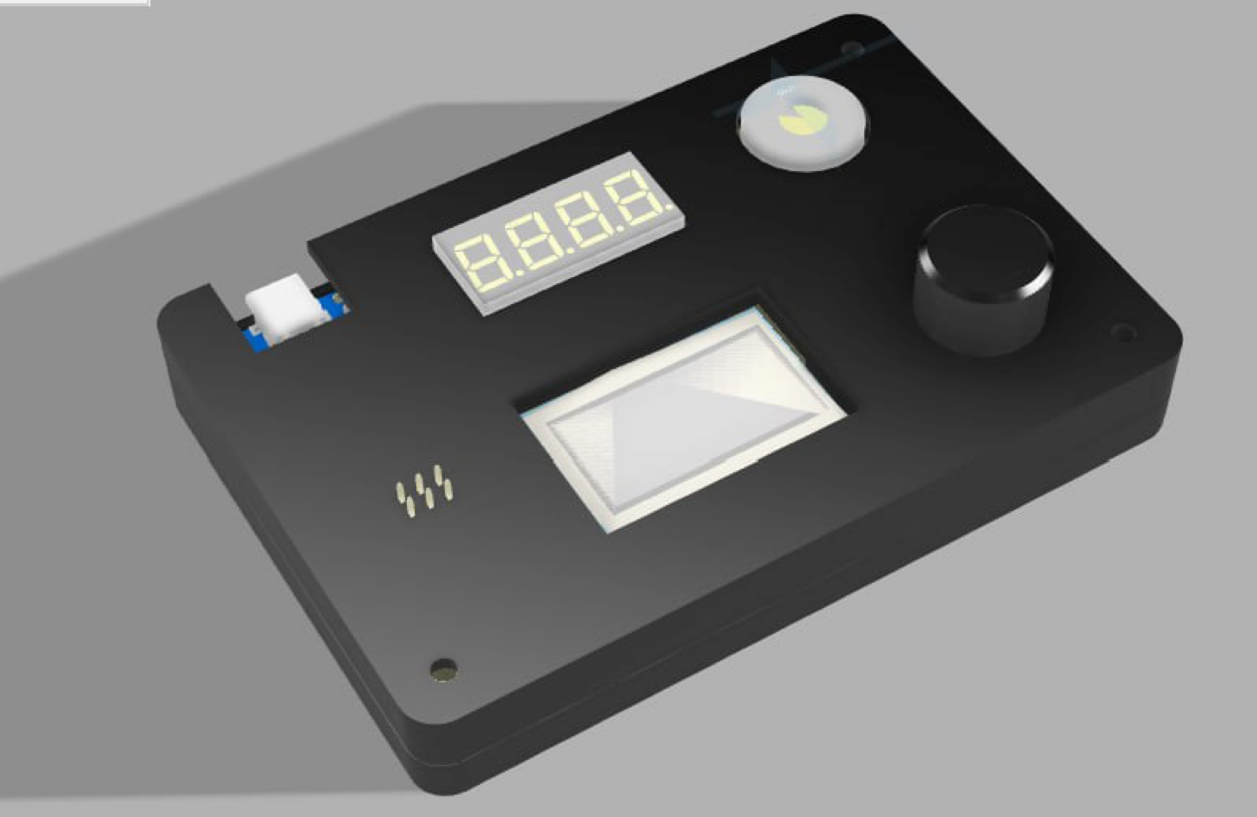


Рисунок 17 – Рендер готовой модели корпус (вид 2)

Разработанный корпус идеально соответствует особенностям устройства «Dino Run». Он обеспечивает надёжную защиту электронных компонентов, удобный доступ к энкодеру, дисплеям и кнопке управления, а также обладает компактными размерами и современным внешним видом. Все конструктивные элементы учтены в модели, и корпус полностью готов к изготовлению методом 3D-печати и последующей сборке с установленной печатной платой и модулями.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта было успешно разработано и реализовано игровое устройство «Dino Run» на базе микроконтроллера Arduino Nano. Устройство полностью соответствует поставленным техническим требованиям: реализован управляемый игровой процесс с отображением текущего счёта на семисегментном дисплее TM1637, графикой на OLED-экране и звуковым сопровождением через пьезоизлучатель. Программа написана с учётом устойчивой работы всех компонентов и обработки пользовательского ввода без блокировок. Также была разработана принципиальная схема, выполнена трассировка печатной платы и спроектирован 3D-корпус, подходящий для изготовления методом 3D-печати. Проект продемонстрировал возможности создания интерактивного микроконтроллерного устройства с визуальной и звуковой обратной связью, а также может быть использован как основа для дальнейших разработок и модификаций.

**Основные результаты проекта:**

1. **Создание прототипа устройства** — разработана виртуальная модель игры *Dino Run* в симуляторе Wokwi для быстрой проверки концепции, затем собран физический прототип на макетной плате c Arduino Nano, TM1637-индикатором, OLED-экраном, энкодером-кнопкой и пьезоизлучателем, что подтвердило правильность аппаратных решений. Дополнительно виртуальный стенд дал возможность безопасно отлаживать прерывания и тайминги, не рискуя повредить реальные компоненты.
2. **Разработка программного обеспечения** — в Arduino IDE написан скетч на C++, реализующий полный игровой цикл: отображение спрайтов динозавра и препятствий на OLED, подсчёт и вывод очков на семисегментный индикатор, управление прыжком по нажатию энкодера, генерацию звуковых эффектов и программное подавление дребезга. Модульная структура кода (функции *drawDino()*, *moveCloud()*, *checkCollision()* и др.) упрощает добавление новых игровых механик без затрагивания уже отлаженных блоков.
3. **Проектирование печатной платы** — в EasyEDA создана принципиальная схема и выполнена односторонняя разводка платы размером ≈ 80 × 60 мм с учётом требований по ширине дорожек и монтажу основных узлов: микроконтроллера, дисплеев, энкодера и зуммера. Все этапы сопровождались автоматическим DRC-контролем, что гарантировало соблюдение технологических норм и позволило сразу подготовить Gerber-файлы к производству.
4. **3D-моделирование корпуса** — в Fusion 360 разработан двухсекционный пластиковый корпус, включающий посадочные места для дисплеев, отверстия под энкодер и крепёж платы, а также акустические решётки для зуммера; конструкция оптимизирована под печать FDM-принтером без поддержек. Дополнительная эргономическая проверка в режиме *In-Canvas Render* показала, что корпус удобно ложится в руку и все органы управления находятся в зоне досягаемости большого пальца.

**Практическая значимость:**

Полученное устройство демонстрирует, как на базе доступных компонентов можно создать полноценную портативную игру с графикой, звуком и счётчиком очков.

Кроме того, проект служит наглядным STEM-примером, объединяющим электронику, программирование и 3D-проектирование в одну учебную задачу.

**Приобретенные компетенции:**

* Программирование микроконтроллеров на C++ с использованием прерываний и сторонних библиотек (U8glib, TM1637Display);

Полученные навыки позволят быстрее прототипировать интерактивные устройства в будущих проектах.

* Работа с аппаратными интерфейсами GPIO и I²C;

Особое внимание уделено правильной конфигурации внутренних подтяжек и согласованию уровней сигнала.

* Разработка принципиальных схем и разводка печатных плат;

Освоены принципы минимизации помех и оптимального размещения компонентов на ограниченной площади.

* 3D-моделирование корпусов для электроники;

Практика работы со сборками и механизмами фиксации значительно сократила время на последующую физическую подгонку деталей.

* Системный подход к созданию встроенных устройств «от идеи до готового изделия».

Комбинирование виртуального и физического прототипирования повысило общую надёжность разработки.

**Возможности дальнейшего развития:**

* Сохранение рекордов во внешней EEPROM с отображением таблицы лучших результатов на OLED-дисплее;

Это повысит реиграбельность и мотивацию игроков.

* Беспроводная передача счёта (Bluetooth / Wi-Fi) для публикации результатов в онлайн-таблице;

Архитектура кода уже предусматривает свободные пины для подключения модулей HC-05 или ESP-01.

* Регулировка уровня сложности вращением энкодера (динамическое изменение скорости speed);

Пользователь сможет выбирать «легкий», «нормальный» и «хард»-режим прямо во время игры.

* Питание от Li-Po-аккумулятора с зарядным модулем для автономной работы;

При этом плата рассчитана на установку JST-коннектора и TP4056.

* Добавление дополнительных кнопок или джойстика для расширенных режимов игры.

Это откроет путь к созданию мини-консоли с несколькими встроенными аркадами.

Цели, поставленные в техническом задании, достигнуты в полном объёме. Созданная *Dino Run*-консоль на базе Arduino демонстрирует эффективное применение микроконтроллерных технологий для решения практических инженерных и учебных задач.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Wokwi [Электронный ресурс]. URL: https://wokwi.com/ (дата обращения: 19.03.2025).

2. EasyEDA [Электронный ресурс]. URL:<https://easyeda.com/> (дата обращения: 15.04.2025).

3. YouTube. «Разводим печатные платы в EasyEDA. Большой гайд» [Электронный ресурс] / видео. URL:<https://www.youtube.com/watch?v=9FS1m_K8aWI> (дата обращения: 15.04.2025).

4. Gyver, A. Уроки Arduino: Библиотеки [Электронный ресурс]. URL:<https://alexgyver.ru/lessons/arduino-libraries/?ysclid=mahepvaezl124763726> (дата обращения: 29.03.2025).

5. YouTube. «Строим 3D модели во Fusion 360. Большой гайд» [Электронный ресурс] / видео. URL:https://www.youtube.com/watch?v=I8-h8mLnexw&t=1927s (дата обращения: 3.05.2025).

**ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД ПРОГРАММЫ**

// A simple T-rex Run game based on the 30DLIS kit

// Developed by Ishani and Arijit Sengupta

// Required libraries

#include <U8glib.h>

#include "TM1637Display.h"

// HW Pins required

#define SW 3 // RotaryEncoder SW Pin

#define DT 4 // RotaryEncoder DT Pin

#define CLK 2 // RotaryEncoder CLK Pin

// Define the display connections pins:

#define DCLK 6

#define DIO 5

// pin 10 drives the buzzer

#define buzzer 10

// In Addition, this sketch uses the I2C Pins for the U8G Panel

// A1+A2

// Create display object of type TM1637Display

// The display shows the current score

TM1637Display OurDisplay = TM1637Display(DCLK, DIO);

// Create array that turns all segments off:

const uint8\_t blank[] = {0x00, 0x00, 0x00, 0x00}; // 0xff is a hexidecimal number whose binary

// representation is all zeros

// Create the oled display object - this shows gameplay

// and welcome/win/loss messages

U8GLIB\_SH1106\_128X64 My\_u8g\_Panel(U8G\_I2C\_OPT\_NONE); // I2C / TWI

/\*

For documentation on u8glib functions:

https://github.com/olikraus/u8glib/wiki/userreference

\*/

// Sounds we use for the hit effects

#define jumpSound 700

#define blahSound 125

#define speedSound 1000

#define DBOUNCE 180

// Game states

#define gameStart 0

#define gameEnd 1

#define gamePlaying 2

volatile int gameStatus = gameStart;

// PRE-SAVED BITMAP CONSTANTS

//20 x 21

static unsigned char dinoJump [] U8G\_PROGMEM = {

0x00, 0xFC, 0x07, 0x00, 0xFE, 0x07, 0x00, 0xEE, 0x0F, 0x00, 0xFE, 0x0F,

0x00, 0xFE, 0x0F, 0x00, 0xFE, 0x0F, 0x00, 0xFE, 0x07, 0x06, 0xFF, 0x03,

0xC3, 0xFF, 0x00, 0xE7, 0xFF, 0x00, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0x00,

0xFF, 0x3F, 0x00, 0xFE, 0x3F, 0x00, 0xFC, 0x1F, 0x00, 0xF8, 0x1F, 0x00,

0xF0, 0x1F, 0x00, 0xF0, 0x0E, 0x00, 0x60, 0x0E, 0x00, 0xE0, 0x0E, 0x00,

0xE0, 0x1E, 0x00,

};

//20 x 21

static unsigned char dinoLeft [] U8G\_PROGMEM = {

0x00, 0xFC, 0x07, 0x00, 0xFE, 0x07, 0x00, 0xEE, 0x0F, 0x00, 0xFE, 0x0F,

0x00, 0xFE, 0x0F, 0x00, 0x7E, 0x08, 0x00, 0x7E, 0x00, 0x06, 0xFF, 0x03,

0x87, 0x3F, 0x00, 0xE7, 0xFF, 0x00, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0x00,

0xFF, 0x3F, 0x00, 0xFE, 0x3F, 0x00, 0xFC, 0x1F, 0x00, 0xF8, 0x1F, 0x00,

0xF0, 0x1F, 0x00, 0xE0, 0x1E, 0x00, 0x60, 0x00, 0x00, 0xE0, 0x00, 0x00,

0xE0, 0x00, 0x00,

};

//20 x 21

static unsigned char dinoRight [] U8G\_PROGMEM = {

0x00, 0xFC, 0x07, 0x00, 0xEE, 0x07, 0x00, 0xE6, 0x0F, 0x00, 0xFE, 0x0F,

0x00, 0xFE, 0x0F, 0x00, 0xFE, 0x0F, 0x00, 0x7C, 0x00, 0x06, 0xFF, 0x03,

0xC3, 0xFF, 0x00, 0xE7, 0xFF, 0x00, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0x00,

0xFF, 0x3F, 0x00, 0xFE, 0x3F, 0x00, 0xFC, 0x1F, 0x00, 0xF8, 0x1F, 0x00,

0xF0, 0x1F, 0x00, 0xF0, 0x0F, 0x00, 0xE0, 0x0E, 0x00, 0x00, 0x0E, 0x00,

0x00, 0x1E, 0x00,

};

//shape 0: 39 x 12

static unsigned char cloud [] U8G\_PROGMEM = {

0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0xBE, 0x03, 0x00, 0x00, 0x00,

0x03, 0x06, 0x00, 0x00, 0x80, 0x01, 0x04, 0x00, 0x00, 0x40, 0x00, 0x1C,

0x00, 0x00, 0x40, 0x00, 0xE4, 0x03, 0x00, 0x18, 0x00, 0x00, 0x02, 0xE0,

0x0F, 0x00, 0x00, 0x0E, 0x30, 0x00, 0x00, 0x00, 0x10, 0x10, 0x00, 0x00,

0x00, 0x20, 0x12, 0x00, 0x00, 0x00, 0x40, 0x03, 0xFF, 0xFF, 0xFF, 0x7F,

};

//shape 1: 10 x 20

static unsigned char oneCactus [] U8G\_PROGMEM = {

0x30, 0x00, 0x78, 0x00, 0x78, 0x00, 0x78, 0x00, 0x78, 0x01, 0xFB, 0x03,

0xFF, 0x03, 0xFF, 0x03, 0xFF, 0x03, 0xFF, 0x03, 0xFF, 0x03, 0xFF, 0x01,

0xFE, 0x00, 0x78, 0x00, 0x78, 0x00, 0x78, 0x00, 0x78, 0x00, 0x78, 0x00,

0x78, 0x00, 0x00, 0x00,

};

//shape 2: 20 x 20

static unsigned char twoCactus [] U8G\_PROGMEM = {

0x30, 0xC0, 0x00, 0x38, 0xE0, 0x00, 0x38, 0xE8, 0x00, 0x38, 0xEC, 0x00,

0x38, 0xED, 0x04, 0xBB, 0xED, 0x0E, 0xBB, 0xED, 0x0E, 0xBB, 0xFD, 0x0E,

0xBB, 0xFD, 0x0E, 0xBB, 0xF9, 0x0E, 0xFF, 0xF1, 0x0F, 0xFF, 0xE0, 0x07,

0x7E, 0xE0, 0x01, 0x38, 0xE0, 0x00, 0x38, 0xE0, 0x00, 0x38, 0xE0, 0x00,

0x38, 0xE0, 0x00, 0x38, 0xE0, 0x00, 0x38, 0xE0, 0x00, 0x00, 0x00, 0x00,

};

//shape 3: 20 x 20

static unsigned char threeCactus [] U8G\_PROGMEM = {

0x00, 0xC0, 0x00, 0x18, 0xC0, 0x01, 0x18, 0xC0, 0x01, 0x58, 0xD8, 0x01,

0x58, 0xFC, 0x01, 0x58, 0xFC, 0x0F, 0x78, 0xDC, 0x0F, 0x7F, 0xFC, 0x0F,

0x3B, 0xFD, 0x0D, 0x1B, 0xF9, 0x0C, 0x5B, 0xF5, 0x0F, 0x5B, 0xC5, 0x07,

0x5F, 0xE7, 0x03, 0xDE, 0xE7, 0x01, 0xD8, 0xC3, 0x01, 0x98, 0xC1, 0x01,

0x18, 0xC1, 0x01, 0x18, 0xC1, 0x01, 0x18, 0xE1, 0x01, 0x00, 0x00, 0x00,

};

//shape 4: 6 x 12

static unsigned char oneCactusSmall [] U8G\_PROGMEM = {

0x0C, 0x0C, 0x3C, 0x3D, 0x2D, 0x3D, 0x1D, 0x0E, 0x0C, 0x0C, 0x0C, 0x0C,

};

//shape 5: 12 x 12

static unsigned char twoCactusSmall [] U8G\_PROGMEM = {

0x0C, 0x03, 0x0C, 0x03, 0x6C, 0x0B, 0x6D, 0x0B, 0x6D, 0x0B, 0xBD, 0x0B,

0x1F, 0x0F, 0x0E, 0x03, 0x0C, 0x03, 0x0C, 0x03, 0x0C, 0x03, 0x0C, 0x03,

};

//shape 6: 17 x 12

static unsigned char threeCactusSmall [] U8G\_PROGMEM = {

0x04, 0x41, 0x00, 0x0C, 0x61, 0x00, 0xFC, 0x79, 0x01, 0xFD, 0x7D, 0x01,

0x7D, 0x6D, 0x01, 0x7D, 0x7D, 0x01, 0xCF, 0xE5, 0x01, 0xCE, 0x67, 0x00,

0x8C, 0x67, 0x00, 0x0C, 0x63, 0x00, 0x0C, 0x61, 0x00, 0x0C, 0x61, 0x00,

};

static unsigned char dinoBlah [] U8G\_PROGMEM = {

0x00, 0xFC, 0x07, 0x00, 0xFE, 0x07, 0x00, 0xC6, 0x0F, 0x00, 0xC6, 0x0F,

0x00, 0xCE, 0x0F, 0x00, 0xFE, 0x0F, 0x00, 0xFE, 0x0F, 0x06, 0xFF, 0x03,

0x87, 0x7F, 0x00, 0xE7, 0xFF, 0x00, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0x00,

0xFF, 0x3F, 0x00, 0xFE, 0x3F, 0x00, 0xFC, 0x1F, 0x00, 0xF8, 0x1F, 0x00,

0xF0, 0x1F, 0x00, 0xF0, 0x0E, 0x00, 0x60, 0x0E, 0x00, 0x60, 0x0E, 0x00,

0xE0, 0x1E, 0x00,

};

static unsigned char gameOver [] U8G\_PROGMEM = {

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0xF8, 0x01, 0x02, 0x0C, 0x0C, 0x3E, 0x00, 0x78, 0x60,

0x30, 0x7C, 0xF0, 0x01, 0x0C, 0x01, 0x07, 0x14, 0x0A, 0x02, 0x00, 0x84,

0x40, 0x10, 0x04, 0x10, 0x02, 0x04, 0x00, 0x05, 0x14, 0x0A, 0x02, 0x00,

0x02, 0x41, 0x10, 0x04, 0x10, 0x02, 0x04, 0x00, 0x05, 0x14, 0x0A, 0x02,

0x00, 0x02, 0xC1, 0x18, 0x04, 0x10, 0x02, 0xC4, 0x81, 0x0D, 0x34, 0x0B,

0x3E, 0x00, 0x02, 0x81, 0x08, 0x7C, 0xF0, 0x01, 0x04, 0x81, 0x08, 0x24,

0x09, 0x02, 0x00, 0x02, 0x81, 0x0D, 0x04, 0x10, 0x01, 0x04, 0x81, 0x0F,

0x64, 0x09, 0x02, 0x00, 0x02, 0x01, 0x05, 0x04, 0x10, 0x02, 0x0C, 0xC1,

0x18, 0xC4, 0x08, 0x02, 0x00, 0x84, 0x00, 0x05, 0x04, 0x10, 0x02, 0xF8,

0x41, 0x10, 0xC4, 0x08, 0x3E, 0x00, 0x78, 0x00, 0x07, 0x7C, 0x10, 0x02,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00,

};

// Other Game attributes

// various variables

int currentStateCLK;

int lastStateCLK;

int MyScore = 0;

int dinoMove = 0;

volatile int jumping = 0;

int cloudx = 128;

int obstacles [2] = {1, 4};

int obstaclex [2] = {128, 200};

int speed = 8;

unsigned long startTime = millis(), curTime;

int lastBeep = 0;

// Display the score on the 7seg display

void ShowScore () {

if (gameStatus == gamePlaying) {

curTime = millis ();

MyScore = (curTime - startTime) \* speed / 1000;

OurDisplay.showNumberDecEx(MyScore);

if (MyScore / 100 > lastBeep) {

tone (buzzer, 1000, 100);

delay (150);

tone (buzzer, 1250, 100);

lastBeep = MyScore / 100;

}

}

}

void StartStopGame () {

static unsigned long last\_interrupt = 0;

if (millis() - last\_interrupt > DBOUNCE) {

if (gameStatus == gamePlaying) {

if (jumping == 0) {

jumping = 1;

tone (buzzer, jumpSound, 100);

}

}

else if (gameStatus == gameStart) gameStatus = gamePlaying;

else gameStatus = gameStart;

}

last\_interrupt = millis(); //note the last time the ISR was called

}

void resetGame () {

MyScore = 0;

startTime = millis();

obstaclex[0] = 128;

obstaclex [1] = 200;

dinoMove = 0;

}

void setup() {

OurDisplay.setBrightness(7);

OurDisplay.clear();

resetGame();

ShowScore ();

pinMode(CLK, INPUT);

pinMode(DT, INPUT);

pinMode(SW, INPUT\_PULLUP);

pinMode(buzzer, OUTPUT);

attachInterrupt(digitalPinToInterrupt(SW), StartStopGame, FALLING);

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loop(void) {

ShowScore();

My\_u8g\_Panel.firstPage();

do {

draw();

} while ( My\_u8g\_Panel.nextPage() );

if (gameStatus == gamePlaying) {

moveDino();

moveCloud();

moveObstacles();

}

}

void moveDino () {

if (jumping == 0) dinoMove = (dinoMove + 1) % 3;

else {

if (jumping == 1) {

dinoMove = dinoMove + 8;

if (dinoMove > 32) jumping = 2;

} else {

dinoMove = dinoMove - 8;

if (dinoMove < 8) {

jumping = 0;

dinoMove = 0;

}

}

}

checkCollision ();

}

void moveCloud () {

cloudx --;

if (cloudx < -38) cloudx = 128;

}

void moveObstacles() {

int obx = obstaclex [0];

obx = obx - speed;

if (obx < -20) {

obstaclex[0] = obstaclex[1];

obstaclex[1] = obstaclex[0] + random(80, 125);

obstacles[0] = obstacles[1];

obstacles[1] = random (1, 6);

}

else {

obstaclex[0] = obx;

obstaclex[1] -= speed;

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void draw(void) {

u8g\_prepare();

if (gameStatus == gamePlaying) {

drawDino ();

drawShape (0, cloudx);

drawObsticles ();

}

else if (gameStatus == gameStart) {

My\_u8g\_Panel.drawStr( 0, 10, "Welcome to");

My\_u8g\_Panel.drawStr( 10, 30, "Dino!!");

My\_u8g\_Panel.drawStr( 0, 50, "Push to begin");

resetGame();

ShowScore();

}

else {

My\_u8g\_Panel.drawXBMP( 14, 12, 100, 15, gameOver);

drawDino ();

drawShape (0, cloudx);

drawObsticles ();

}

}

void drawDino (void) {

if (gameStatus == gameEnd) {

My\_u8g\_Panel.drawXBMP( 0, 43 - dinoMove, 20, 21, dinoBlah);

return;

}

switch (dinoMove) {

case -1: My\_u8g\_Panel.drawXBMP( 0, 43, 20, 21, dinoBlah); break;

case 0: My\_u8g\_Panel.drawXBMP( 0, 43, 20, 21, dinoJump); break;

case 1: My\_u8g\_Panel.drawXBMP( 0, 43, 20, 21, dinoLeft); break;

case 2: My\_u8g\_Panel.drawXBMP( 0, 43, 20, 21, dinoRight); break;

default: My\_u8g\_Panel.drawXBMP( 0, 43 - dinoMove, 20, 21, dinoJump); break;

}

}

void drawShape (int shape, int x) {

switch (shape) {

case 0: My\_u8g\_Panel.drawXBMP( x, 5, 39, 12, cloud); break;

case 1: My\_u8g\_Panel.drawXBMP( x, 44, 10, 20, oneCactus); break;

case 2: My\_u8g\_Panel.drawXBMP( x, 44, 20, 20, twoCactus); break;

case 3: My\_u8g\_Panel.drawXBMP( x, 44, 20, 20, threeCactus); break;

case 4: My\_u8g\_Panel.drawXBMP( x, 52, 6, 12, oneCactusSmall); break;

case 5: My\_u8g\_Panel.drawXBMP( x, 52, 12, 12, twoCactusSmall); break;

case 6: My\_u8g\_Panel.drawXBMP( x, 52, 17, 12, threeCactusSmall); break;

}

}

void checkCollision () {

int obx = obstaclex [0];

int obw, obh;

switch (obstacles[0]) {

case 0: obw = 39; obh = 10; break;

case 1: obw = 10; obh = 20; break;

case 2: obw = 17; obh = 20; break;

case 3: obw = 17; obh = 20; break;

case 4: obw = 6; obh = 12; break;

case 5: obw = 12; obh = 12; break;

case 6: obw = 17; obh = 12; break;

}

if (obx > 15 || obx + obw < 5 || dinoMove > obh - 3) {}

else {

gameStatus = gameEnd;

tone (buzzer, 125, 100);

delay(150);

tone (buzzer, 125, 100);

}

}

void drawObsticles () {

drawShape (obstacles[0], obstaclex[0]);

drawShape (obstacles[1], obstaclex[1]);

}

void u8g\_prepare(void) {

My\_u8g\_Panel.setFont(u8g\_font\_6x10);

My\_u8g\_Panel.setFontRefHeightExtendedText();

My\_u8g\_Panel.setDefaultForegroundColor();

My\_u8g\_Panel.setFontPosTop();

}