Министерство образования и науки Российской Федерации

федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет \_\_информационных технологий\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра \_\_\_прикладной математики\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Специальность (направление, профиль)\_\_ПИ\_\_\_\_\_\_\_\_\_\_\_\_\_

Курсовой проект

защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.С. Троицкий

(подпись руководителя проекта)(инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Курсовой проект

Разработка 2D игры «CursGolf» в жанре минигольф с элементами головоломки

(тема курсового проекта)

Пояснительная записка

по дисциплине \_\_Программирование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_КП 09.03.04.21.000 ПЗ\_\_\_\_\_\_\_

(обозначение документа)

Студенты группы ПИ-32 Балдин\_А.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_23.09.2024

(фамилия, имя, отчество) (подпись) (дата)

Студенты группы ПИ-32 Кучковский\_С.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_23.09.2024

(фамилия, имя, отчество) (подпись) (дата)

Руководитель проекта преподаватель.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.С. Троицкий

(должность, ученое звание) (подпись) (инициалы, фамилия)

БАРНАУЛ 2025

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный технический университет

имени И.И. Ползунова»

Факультет информационных технологий

Кафедра «Прикладная математика»

З А Д А Н И Е

на курсовой проект по дисциплине «Программирование»

студентам группы ПИ-32 Балдину Андрею Андреевичу, Кучковскому Станиславу Владимировичу

Тема курсового проекта: «Разработка 2D игры «CursGolf» в жанре минигольф с элементами головоломки »

Календарный план работы:

|  |  |  |
| --- | --- | --- |
| № этапа | Содержание этапа | Недели семестра |
| 1 | Получение задания | 1 |
| 2 | Постановка задачи. Работа с документацией | 2 |
| 3 | Проектирование программы | 3-4 |
| 4 | Реализация программы | 5-13 |
| 5 | Оформление пояснительной записки | 14 |
| 6 | Защита курсового проекта | 15-16 |

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.С. Троицкий, доцент

подпись

Дата выдачи задания «23» сентября 2024 г.

число месяц год

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Балдин А.А.

Подпись

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кучковский С.В.

подпись

**Содержание**

Введение4

1. Обзор предметной области и постановка задачи 5

1.1 Обзор предметной области...............................................................................................5

1.2 Постановка задачи5

2. Проектирование9

2.1 Укрупненный алгоритм решения9

2.2 Структура данных9

3. Реализация10

3.1 Выбор средств реализации10

3.2 Структура программы10

3.3 Состав программы11

Заключение13

Список использованных источников14

Приложение А15

Приложение Б77

Введение

Разработка компьютерных игр представляет собой одну из самых динамично развивающихся отраслей информационных технологий. Игры находят широкое применение не только в индустрии развлечений, но и в образовании, маркетинге и даже в медицинских исследованиях. Особый интерес вызывает жанр казуальных игр, таких как минигольф, которые отличаются простотой освоения, увлекательностью и доступностью для широкой аудитории. Этот проект ставит перед собой одну из задач создания интересного и интуитивно понятного игрового процесса, который позволит игрокам испытать элементы стратегического мышления и точности в увлекательной игровой форме.

В процессе работы над проектом будут рассмотрены ключевые аспекты разработки компьютерных игр, включая дизайн уровней, реализацию физики движения объектов, управление пользователем и создание визуального оформления. Кроме того, особое внимание будет уделено выбору инструментов и технологий для разработки, а также методам оптимизации игрового процесса.

Цель данной работы: изучение принципов объектно-ориентированного программирования и обучение их применению на реальной задаче.

Главная задача: разработать 2D игру в жанре минигольф с пиксельной графикой и элементами головоломки с помощью языка c++ с библиотекой для отрисовки графики SDL, написать прототип игры на java с использованием библиотеки LWJGL (Lightweight Java Game Library).

1 Обзор предметной области и постановка задачи

* 1. **Обзор предметной области**

Объектно-ориентированное программирование (ООП) — это подход, при котором программа рассматривается как набор объектов, взаимодействующих друг с другом. У каждого есть свойства и поведение.

Объектно-ориентированное программирование является парадигмой разработки программного продукта с использованием трех принципов: инкапсуляция, полиморфизм, наследование.

Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Инкапсуляция или «сокрытие информации» предполагает, что пользователь обращается к созданному в программе объекту через открытый интерфейс взаимодействия, а не напрямую, детали от пользователя скрыты.

Полиморфизм — это свойство программного кода изменять свое поведение в зависимости от ситуации, возникающей при выполнении программы. Полиморфизм позволяет повторно использовать уже написанный ранее код, что повышает эффективность разработки.

Наследование даёт возможность объекту (классу) использовать переменные и методы другого объекта (класса) как свои собственные, что также повышает эффективность разработки.

Без следования любому из этих принципов модель не будет объектно-ориентированной.

1.2 Постановка задачи

Разработать 2D игру в жанре минигольф используя парадигму объектно-ориентированного программирования под платформу Windows 10 или выше, на языке С++ с использованием библиотеки SDL2, написать прототип игры на java с использованием библиотеки LWJGL (Lightweight Java Game Library).

Игра должна включать в себя 10 уровней. Должно быть разработано меню с выбором «Game», «Info» и «Exit». В пункте «Game» пользователь начинает игру с первого уровня. В пункте «Info» пользователь может посмотреть информацию об игре. У игрока в любой момент времени должна быть возможность выйти в главное меню и начать игру заново. В пункте «Records» содержится информация о лучших результатах игрока за каждый уровень. Лучший результат – наименьшее количество ударов по мячу. Пользователь должен управлять мячом с помощью мыши. В игре необходимо правильно составить логику взаимодействия пользователя и мяча.

**Схема уровней для игры:**

Изображение выглядит как диаграмма, План, Технический чертеж, зарисовка

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 Схема первого игрового уровня

На схеме отмечены: мячик, которым управляет игрок, лунка с флажком, в которую должен попасть мячик для перехода на следующий уровень, а также различные текстуры, имеющие определенные свойства. Песок (встречается на других уровнях) и земля замедляют шарик. Через серые блоки запрещено проходить, при столкновении с ними шарик отскакивает, имитируется физика мяча. По краям карты расположено время, отсчет ведется от начала уровня, количество ударов по мячу (Strokes), ограничитель ударов (16), а также лучший результат на этом уровне. Для выхода в главное меню нужно нажать ESC.

Изображение выглядит как диаграмма, зарисовка, Технический чертеж, План

Контент, сгенерированный ИИ, может содержать ошибки.



Рисунок 2 Схема второго уровня с песком

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 Схема меню

Меню состоит из четырех опций: «Game», «Records», «Info» , «Exit». Текущая выбранная строка выделена зеленым цветом. Для перемещения по опциям используются стрелочки на клавиатуре. Чтобы выбрать опцию, нажмите ENTER. Выбор на первой опции начинает игру с первого уровня. При выборе второй опции выводится окно, содержащая лучшие результаты игрока за каждый уровень. Третья опция выводит основную информацию об игре. Для выхода из игры выберите «Exit».

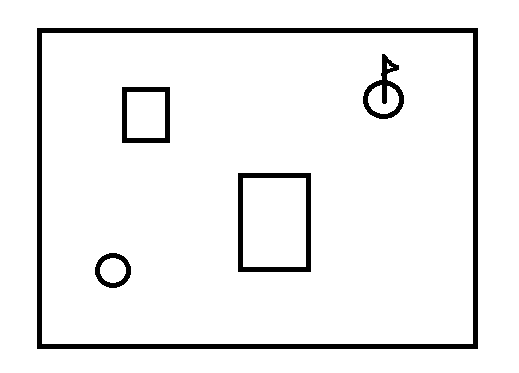
Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 Информация об игре

Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5 Рекорды

Лунка с флажком



Мячик



Рисунок 6 Прототип на java

**2 Проектирование**

**2.1 Укрупненный алгоритм решения**

Программа работает в диалоговом режиме. При исполнении программы пользователь видит перед собой меню с возможностью выбора из 4 вариантов: «Играть», «Рекорды», «Информация» и «Выход». При нажатии кнопки «Играть» пользователь перейдет на 1ый уровень. Уровни идут по порядку, для получения доступа к каждому новому уровню необходимо пройти предыдущий. Информация о лучшем результате хранится в разделе «Рекорды». При нажатии на кнопку «Информация» пользователь увидит основную информацию об игре: управление, описание. Из любого выше описанного окна меню можно вернуться в основное меню, нажав кнопку «ESC» на клавиатуре.

2.2 Структура данных

## Каждый отдельный модуль реализован с помощью классов. Графический интерфейс и работа с объектами реализовывалась при помощи библиотеки SDL2. При реализации игрового движка использовался паттерн **Entity-Component-System.** Для упрощения работы с реализацией физики мяча была написана своя структура данных: Vector2D, которая представляет собой вектор на плоскости. Помимо этого активно использовались массивы(статические и динамические), битсеты, словари.

**Распределение задач:**

Разработка классов для правильного взаимодействия пользователя с мячом, для физики мяча, для создания полноценного игрового движка; оформление курсовой работы

– Балдин А.А и Кучковский С.В.

Разработка алгоритма добавления уровней и создание уровней 6-10

– Кучковский С.В.

Разработка взаимодействия пользователя с меню, создание уровней 1-5

– Балдин А.А

**3 Реализация**

**3.1 Выбор средств реализации**

Данное ПО реализовано на языке C++ с использованием среды разработки Microsoft Visual Studio 2022 и библиотеки SDL2 для отрисовки объектов игры, включая дополнительные библиотеки SDL2\_ttf для шрифта, SDL2\_mixer звуков. Для реализации используется парадигма объектно-ориентированного программирования. Прототип реализован на языке java с использованием IntelliJ IDEA Community Edition 2024.2.1 и библиотеки LWJGL для отрисовки объектов игры.

3.2 Структура программы

В программе реализованы следующие модули и взаимодействие между ними, изображенные на рисунке 14.

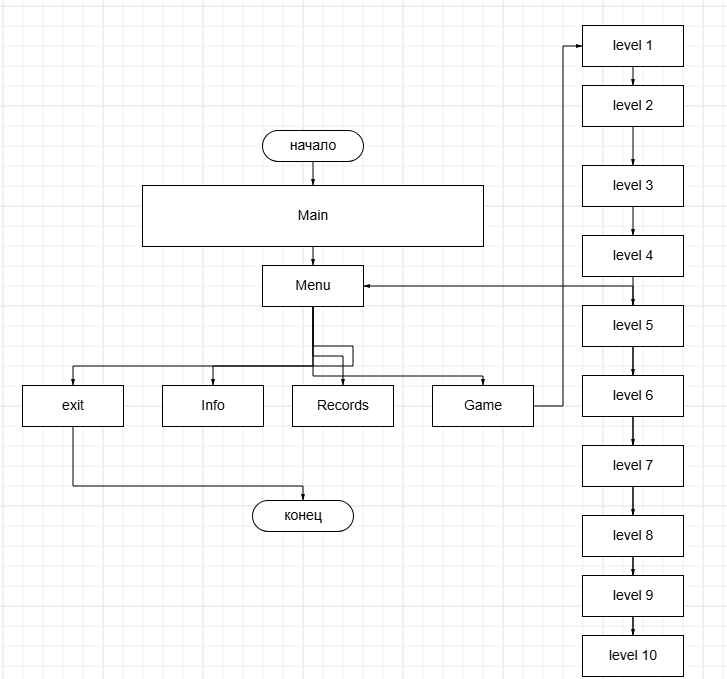




Рисунок 7 Структура программы в виде схемы и иерархии модулей

**3.3 Состав программы**

**Модули (функции) программы:**

**Entity-Component System (система сущностей и компонентов):**

**BallMechanic –** компонент, реализующий физику мяча.

**ColiderComponent –** компонент, реализующий механику взаимодействия объектов. При столкновении мяча и любого другого объекта (имеющий этот компонент), мяч отскакивает.

**Components.h –** заголовочный файл, содержащий в себе все компоненты

**ECS.h –** система, в которой описаны: класс Сущности, базовый класс Компонентов, а также Менеджер сущностей (класс, управляющий сущностями).

**ECS.cpp –** файл, в котором описана реализация некоторых методов, принадлежащих ECS.

**SpriteComponent.h –** компонент, реализующий работу со спрайтами (графические объекты) в игре.

**TileComponent.h –** компонент,работающий с мелкими блоками, из которых составлена карта.

**TransformComponent.h –** компонент, объединяющий в себе логику движения объектов.

**Модули, взаимодействующие с ECS:**

**Сollision.cpp и Collision.h –** модуль, описывающий наличия коллизий, т.е. взаимодействие между объектами, а если точнее, то их столкновение.

**Game.cpp и Game.h –** модуль, реализующий поэтапную логику игры: инициализацию объектов, рендер, обновление, переход на новый уровень и др.

**InfoWindow.cpp и InfoWindow.h –** модуль, который создает отдельный блок в меню с описанием программного продукта

**Main.cpp –** файл, в котором описана логика выполнения программы.

**Map.cpp и Map.h –** модуль, в котором описаны методы для работы с картой: создание, отображение, инициализация. Карта представляет собой отдельный текстовый файл, где записаны 2 двумерных массива чисел. Первый массив отвечает за загрузку текстур, 2ой – за инициализацию препятствий, за которые шарик не может заходить. Каждой цифре соответствует своя текстура. Все tileset’ы доступны в директории assets главного каталога.

**Menu.cpp и Menu.h –** модуль, отвечающий за представление меню.

**TextureManager.cpp и TextureManager.h –** модуль, отвечающий за загрузку и прорисовку текстур.

**Vector2D.cpp и Vector2D.h –** модуль, реализующий структуру данных двумерного вектора и операции с векторами.

**Прототип на java:**

**GolfGame.java** – модуль, реализующий весь прототип основной игры для языка java

Заключение

Был создан игровой движок, соответствующий проекту и актуальный на рынке игровой индустрии. На основе собственного движка написана игра. Реализованы уровни с различными механиками и сценариями игры.

В процессе выполнения задания была подробно изучена парадигма объектно-ориентированного программирования на языке программирования C++.

Преимущества приложения:

* Легкий интерфейс;
* Работает без доступа к сети Интернет;
* Не требует высокой производительности устройства;
* Не требует установки;
* Есть звуковое сопровождение;
* Простое интуитивно понятное управление;
* Интересный геймплей;

Недочёты приложения:

* Физика мяча требует доработки;
* Отсутствие настроек и параметров;
* Возможны некоторые мелкие баги;
* Отсутствие паузы
* Нет большого количества уровней
* Коллизии объектов иногда неисправны

Список использованных источников

1. Сайт Stack Overflow [Электронный ресурс]. Режим доступа: https://stackoverflow.com/, свободный.
2. Сайт GitHub [Электронный ресурс]. Режим доступа: https://github.com/, свободный.
3. Сайт Habr [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles, свободный
4. Сайт SDL2 [Электронный ресурс]. Режим доступа: https://www.libsdl.org/, свободный
5. Сайт Microsoft learn [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/cpp/?view=msvc-170, свободный
6. Егорова Е. В. Программирование на языке СИ. Учебное пособие/Алт. Госуд. Технич. Ун-т им. И.И.Ползунова. - Барнаул: 2013.- 184с.

Приложение А

Исходный код программы

**BallMechanic:**

#pragma once

#include "../Game.h"

#include "ECS.h"

#include "Components.h"

#include <SDL\_ttf.h>

#include <sstream>

#include "SDL\_mixer.h"

#include "SDL.h"

class Particle {

public:

Vector2D position;

Vector2D velocity;

float lifetime; // Время жизни частицы в секундах

float size; // Размер частицы

Particle(float x, float y, float vx, float vy, float lifetime, float size)

: position(x, y), velocity(vx, vy), lifetime(lifetime), size(size) {}

void update(float deltaTime) {

position.x += velocity.x \* deltaTime;

position.y += velocity.y \* deltaTime;

lifetime -= deltaTime;

}

bool isDead() const {

return lifetime <= 0;

}

};

class BallMechanic : public Component

{

public:

TransformComponent\* transform;

Vector2D target;

Vector2D launchedVelocity;

Vector2D initialMousePos;

Vector2D currentMousePos;

Mix\_Chunk\* swingSound = nullptr;

bool canMove = true;

bool playedSwingFx = true;

int index;

int strokes = 0;

int dirX = 1;

int dirY = 1;

float velocityX;

float velocityY;

float velocityLength;

bool strokesLimitFlag = false;

bool win = false;

float friction = 0.001;

bool isMoving = false; // Флаг для проверки, что мяч двигается

float stopThreshold = 0.005f;

float stopConst = 0.002f;

std::vector<std::unique\_ptr<Particle>> particles;

std::string to\_string(int value) {

std::stringstream ss;

ss << value;

return ss.str();

}

template <typename T>

T clamp(T value, T min, T max) {

if (value < min) return min;

if (value > max) return max;

return value;

}

void init() override

{

transform = &entity->getComponent<TransformComponent>();

swingSound = Mix\_LoadWAV("assets/sfx/swing.mp3");

if (!swingSound) {

std::cerr << "Failed to load swing sound! SDL\_mixer Error: " << Mix\_GetError() << std::endl;

}

std::cout << "Added BALL MECHANIC COMPONENT\n";

}

Vector2D& getVelocity()

{

return transform->velocity;

}

Vector2D& getInitialMousePos()

{

return initialMousePos;

}

void setVelocity(float x, float y)

{

transform->velocity.x = x;

transform->velocity.y = y;

}

void setInitialMousePos(float x, float y)

{

initialMousePos.x = x;

initialMousePos.y = y;

}

void setCurrentMousePos(float x, float y)

{

currentMousePos.x = x;

currentMousePos.y = y;

}

// вектор скорости

float getDistance(Vector2D a, Vector2D b) {

return SDL\_sqrt(SDL\_pow(abs(a.x - b.x), 2) + SDL\_pow(abs(a.y - b.y), 2));

}

void setWin(bool p\_win)

{

Game::win = p\_win;

}

void update() override

{

float deltaTime = 1.0f / 60.0f;

// Проверяем, двигается ли мяч

if (SDL\_sqrt(transform->velocity.x \* transform->velocity.x + transform->velocity.y \* transform->velocity.y) > stopThreshold)

{

isMoving = true;

}

else

{

isMoving = false;

}

if (isMoving) {

// Генерация новых частиц позади мяча

float speed = SDL\_sqrt(transform->velocity.x \* transform->velocity.x + transform->velocity.y \* transform->velocity.y);

float particleSize = clamp(8.0f - speed \* 1.5f, 2.0f, 8.0f); // Ширина частицы уменьшается с увеличением скорости

particles.emplace\_back(std::make\_unique<Particle>(

transform->position.x + 13, // Центр мяча

transform->position.y + 13,

-transform->velocity.x \* 0.5f, // Немного медленнее, чем мяч

-transform->velocity.y \* 0.5f,

0.5f, // Время жизни частицы

particleSize

));

}

// Обновляем частицы

for (auto& particle : particles) {

particle->update(deltaTime);

}

// Удаляем мертвые частицы

particles.erase(

std::remove\_if(particles.begin(), particles.end(), [](const std::unique\_ptr<Particle>& p) { return p->isDead(); }),

particles.end()

);

if (!isMoving) // Разрешаем новый удар только если мяч не двигается

{

int mouseX = 0;

int mouseY = 0;

switch (Game::event.type)

{

case SDL\_MOUSEBUTTONDOWN:

if (Game::event.button.button == SDL\_BUTTON\_LEFT)

{

// Находим положение относительно рабочего стола

SDL\_GetMouseState(&mouseX, &mouseY);

// Присваиваем эти координаты initialMousePos.x и initialMousePos.y

setInitialMousePos(mouseX, mouseY);

std::cout << mouseX << " ; " << mouseY << std::endl;

}

break;

case SDL\_MOUSEMOTION:

if (SDL\_GetMouseState(nullptr, nullptr) & SDL\_BUTTON(SDL\_BUTTON\_LEFT))

{

// Обновляем положение мыши и рассчитываем скорость

SDL\_GetMouseState(&mouseX, &mouseY);

velocityX = (mouseX - getInitialMousePos().x) / -60.0f;

velocityY = (mouseY - getInitialMousePos().y) / -60.0f;

// Ограничиваем значения скорости

velocityX = clamp(velocityX, -5.0f, 5.0f);

velocityY = clamp(velocityY, -5.0f, 5.0f);

}

break;

case SDL\_MOUSEBUTTONUP:

if (Game::event.button.button == SDL\_BUTTON\_LEFT)

{

SDL\_GetMouseState(&mouseX, &mouseY);

// Задаем вектор скорости

setVelocity((mouseX - getInitialMousePos().x) / -60, (mouseY - getInitialMousePos().y) / -60);

float velocityX1 = transform->velocity.x;

float velocityY1 = transform->velocity.y;

float velocityMagnitude = sqrt(velocityX1 \* velocityX1 + velocityY1 \* velocityY1);

// Устанавливаем максимальную скорость

const float maxVelocity = 5.0f;

if (velocityMagnitude > maxVelocity) {

float normalizationFactor = maxVelocity / velocityMagnitude;

transform->velocity.x \*= normalizationFactor;

transform->velocity.y \*= normalizationFactor;

}

// Находим модуль скорости через теорему Пифагора (v0)

transform->speed = SDL\_sqrt(SDL\_pow(abs(getVelocity().x), 2) + SDL\_pow(abs(getVelocity().y), 2));

// Замедляем шарик при помощи отрицательного ускорения

transform->acceleration.x = -transform->velocity.x \* stopConst;

transform->acceleration.y = -transform->velocity.y \* stopConst;

// Увеличиваем количество ударов

strokes++;

if (swingSound) {

Mix\_PlayChannel(-1, swingSound, 0);

}

}

break;

}

}

}

Vector2D getPos()

{

return transform->position;

}

void draw() override

{

int mouseX = 0;

int mouseY = 0;

if (SDL\_GetMouseState(&mouseX, &mouseY))

{

// Рассчитать конец линии

float dx = -1\*(mouseX - getInitialMousePos().x);

float dy = -1\*(mouseY - getInitialMousePos().y);

// Нормализация вектора

float length = SDL\_sqrt(dx \* dx + dy \* dy);

if (length > 0)

{

dx /= length;

dy /= length;

}

velocityLength = SDL\_sqrt(velocityX \* velocityX + velocityY \* velocityY);

// Длина линии

// Минимальная и максимальная длина линии

float minLength = 16.0f;

float maxLength = 80.0f;

// Длина линии изменяется в зависимости от скорости мяча

float lineLength = minLength + (velocityLength \* 20.0f); // Увеличиваем длину в зависимости от скорости

// Ограничиваем длину линии

if (lineLength > maxLength) {

lineLength = maxLength;

}

// Параметры наконечника стрелки

float arrowLength = 15.0f; // Длина наконечника стрелки

float arrowWidth = 10.0f; // Ширина наконечника стрелки

// Радиус мяча

float ballRadius = 16.0f; // Половина размера мяча (32 / 2)

// Начало линии — окружность мяча

float startX = transform->position.x + ballRadius + dx \* ballRadius;

float startY = transform->position.y + ballRadius + dy \* ballRadius;

// Конец основной линии (середина основания стрелки)

float lineEndX = startX + dx \* (lineLength - arrowLength);

float lineEndY = startY + dy \* (lineLength - arrowLength);

// Острие стрелки

float arrowTipX = startX + dx \* lineLength;

float arrowTipY = startY + dy \* lineLength;

// Левая точка основания наконечника стрелки

float leftBaseX = lineEndX - arrowWidth / 2 \* dy;

float leftBaseY = lineEndY + arrowWidth / 2 \* dx;

// Правая точка основания наконечника стрелки

float rightBaseX = lineEndX + arrowWidth / 2 \* dy;

float rightBaseY = lineEndY - arrowWidth / 2 \* dx;

// Установить черный цвет

SDL\_SetRenderDrawColor(Game::renderer, 0, 0, 0, 255);

// Рисуем основную линию

SDL\_RenderDrawLine(Game::renderer, static\_cast<int>(startX), static\_cast<int>(startY),

static\_cast<int>(lineEndX), static\_cast<int>(lineEndY));

// Рисуем треугольник (наконечник стрелки)

SDL\_RenderDrawLine(Game::renderer, static\_cast<int>(arrowTipX), static\_cast<int>(arrowTipY),

static\_cast<int>(leftBaseX), static\_cast<int>(leftBaseY));

SDL\_RenderDrawLine(Game::renderer, static\_cast<int>(arrowTipX), static\_cast<int>(arrowTipY),

static\_cast<int>(rightBaseX), static\_cast<int>(rightBaseY));

SDL\_RenderDrawLine(Game::renderer, static\_cast<int>(leftBaseX), static\_cast<int>(leftBaseY),

static\_cast<int>(rightBaseX), static\_cast<int>(rightBaseY));

}

for (const auto& particle : particles) {

SDL\_SetRenderDrawColor(Game::renderer, 255, 255, 255, 255); // Белый цвет

SDL\_Rect rect = { static\_cast<int>(particle->position.x), static\_cast<int>(particle->position.y), static\_cast<int>(particle->size), static\_cast<int>(particle->size) };

SDL\_RenderFillRect(Game::renderer, &rect);

}

}

bool isWin() const

{

return Game::win; // Проверяем глобальную переменную

}

};

**ColiderComponent :**

#pragma once

#include <string>

#include "SDL.h"

#include "ECS.h"

#include "Components.h"

#include "../TextureManager.h"

class ColliderComponent : public Component

{

public:

SDL\_Rect collider;

std::string tag;

SDL\_Texture\* tex;

SDL\_Rect srcR, destR;

TransformComponent\* transform;

ColliderComponent(std::string t)

{

tag = t;

}

ColliderComponent(std::string t, int xpos, int ypos, int size)

{

tag = t;

collider.x = xpos;

collider.y = ypos;

collider.h = collider.w = size;

}

void init() override

{

if (!entity->hasComponent<TransformComponent>())

{

entity->addComponent<TransformComponent>();

}

transform = &entity->getComponent<TransformComponent>();

//tex = TextureManager::LoadTexture("assets/ColTex.png");

srcR = { 0, 0, 32, 32 };

destR = { 0, 0, 32, 32 };

std::cout << "Added COLLIDER COMPONENT\n";

}

void draw() override

{

TextureManager::Draw(tex, srcR, destR);

}

void update() override

{

if (tag != "terrain" && tag != "UPPDOWNER" && tag != "LEFTRIGHTER"

&& tag != "corUPLEFT" && tag != "corUPRIGHT" && tag != "corDOWNLEFT"

&& tag != "corDOWNRIGHT" && tag != "dirt" && tag != "sand")

{

collider.x = static\_cast<int>(transform->position.x);

collider.y = static\_cast<int>(transform->position.y);

collider.w = transform->width \* transform->scale;

collider.h = transform->height \* transform->scale;

}

destR.x = collider.x;

destR.y = collider.y;

if (tag == "hole") {

collider.x = static\_cast<int>(transform->position.x)+15;

collider.y = static\_cast<int>(transform->position.y)+15;

collider.w = (transform->width \* transform->scale)-30;

collider.h = (transform->height \* transform->scale)-30;

}

}

};

**Components.h:**

#pragma once

#include "ECS.h"

#include "TransformComponent.h"

#include "SpriteComponent.h"

#include "BallMechanic.h"

#include "ColiderComponent.h"

#include "TileComponent.h"

**ECS.cpp :**

#include "ECS.h"

void Entity::addGroup(Group mGroup)

{

groupBitset[mGroup] = true;

manager.AddToGroup(this, mGroup);

}

Manager::Manager()

{

}

**ECS.h:**

#pragma once

#include <iostream>

#include <vector>

#include <memory>

#include <algorithm>

#include <bitset>

#include <array>

class Component;

class Entity;

class Manager;

using ComponentID = std::size\_t; // like unsigned int

using Group = std::size\_t;

inline ComponentID getNewComponentTypeID()

{

static ComponentID lastID = 0u;

return lastID++;

}

template <typename T> inline ComponentID getComponentTypeID() noexcept

{

static ComponentID typeID = getNewComponentTypeID();

return typeID;

}

constexpr std::size\_t maxComponents = 32;

constexpr std::size\_t maxGroups = 32;

using ComponentBitSet = std::bitset<maxComponents>; // fixed-size sequence of N bits

using GroupBitset = std::bitset<maxGroups>;

using ComponentArray = std::array<Component\*, maxComponents>; // create an array of component pointers

class Component

{

public:

Entity \*entity;

virtual void init() {}

virtual void update() {}

virtual void draw() {}

virtual ~Component() {}

};

class Entity {

private:

Manager& manager;

GroupBitset groupBitset;

public:

Entity(Manager& mManager) : manager(mManager) {}

void update() {

for (auto& c : components) c->update();

}

void draw() {

for (auto& c : components) c->draw();

}

bool isActive() { return active; }

void destroy() { active = false; }

void revive() { active = true; }

bool hasGroup(Group mGroup)

{

return groupBitset[mGroup];

}

void addGroup(Group mGroup);

void delGroup(Group mGroup)

{

groupBitset[mGroup] = false;

}

template <typename T> bool hasComponent() const {

return componentBitSet[getComponentTypeID<T>()];

}

// компонент сущности добавляется в массив компонентов

template <typename T, typename... TArgs>

T& addComponent(TArgs&&... mArgs)

{

T\* c(new T(std::forward<TArgs>(mArgs)...));

c->entity = this;

std::unique\_ptr<Component> uPtr{ c };

components.emplace\_back(std::move(uPtr));

componentArray[getComponentTypeID<T>()]= c;

componentBitSet[getComponentTypeID<T>()] = true;

c->init();

return \*c;

}

// берем компонент сущности

template<typename T> T& getComponent() const {

auto ptr(componentArray[getComponentTypeID<T>()]);

return \*static\_cast<T\*>(ptr);

}

private:

bool active = true;

std::vector<std::unique\_ptr<Component>> components;

ComponentArray componentArray;

ComponentBitSet componentBitSet;

};

class Manager

{

private:

std::vector<std::unique\_ptr<Entity>> entities;

std::array<std::vector<Entity\*>, maxGroups> groupedEntities;

public:

Manager();

void update()

{

for (auto& e : entities) e->update();

}

void draw()

{

for (auto& e : entities) e->draw();

}

void refresh()

{

// удаляем Entities из группы

for (auto i(0u); i < maxGroups; i++)

{

auto& v(groupedEntities[i]);

v.erase(

std::remove\_if(std::begin(v), std::end(v),

[i](Entity\* mEntity)

{

return !mEntity->isActive() || !mEntity->hasGroup(i);

}),

std::end(v));

}

entities.erase(std::remove\_if(std::begin(entities), std::end(entities),

[](const std::unique\_ptr<Entity>& mEntity)

{

return !mEntity->isActive();

}),

std::end(entities));

}

void AddToGroup(Entity\* mEntity, Group mGroup)

{

groupedEntities[mGroup].emplace\_back(mEntity);

}

std::vector<Entity\*>& getGroup(Group mGroup)

{

return groupedEntities[mGroup];

}

Entity& addEntity()

{

Entity\* e = new Entity(\*this);

std::unique\_ptr<Entity> uPtr{ e };

entities.emplace\_back(std::move(uPtr));

return \*e;

}

};

**SpriteComponent.h :**

#pragma once

#include "Components.h"

#include "SDL.h"

#include "../TextureManager.h"

class SpriteComponent : public Component

{

private:

TransformComponent\* transform;

SDL\_Texture\* texture;

SDL\_Rect srcRect, destRect;

public:

SpriteComponent() = default;

SpriteComponent(const char\* path)

{

setTex(path);

}

~SpriteComponent() {

SDL\_DestroyTexture(texture);

}

void init() override

{

transform = &entity->getComponent<TransformComponent>();

srcRect.x = srcRect.y = 0;

srcRect.w = transform->width;

srcRect.h = transform->height;

destRect.w = destRect.h = 32;

}

void setTex(const char\* path)

{

texture = TextureManager::LoadTexture(path);

std::cout << "Added TEXTURE COMPONENT\n";

}

void update() override

{

destRect.x = static\_cast<int>(transform->position.x);

destRect.y = static\_cast<int>(transform->position.y);

destRect.w = transform->width \* transform->scale;

destRect.h = transform->height \* transform->scale;

}

void draw() override

{

TextureManager::Draw(texture, srcRect, destRect);

}

};

**TileComponent.h :**

#pragma once

#include "ECS.h"

#include "SDL.h"

class TileComponent : public Component

{

public:

SDL\_Texture\* texture;

SDL\_Rect srcRect, destRect;

TileComponent() = default;

~TileComponent()

{

SDL\_DestroyTexture(texture);

}

TileComponent(int srcX, int srcY, int xpos, int ypos, int tsize, int tscale, const char\* path)

{

texture = TextureManager::LoadTexture(path);

srcRect.x = srcX;

srcRect.y = srcY;

srcRect.w = srcRect.h = tsize;

destRect.x = xpos;

destRect.y = ypos;

destRect.w = destRect.h = tsize \* tscale;

}

void draw() override

{

TextureManager::Draw(texture, srcRect, destRect);

}

};

**TransformComponent.h:**

#pragma once

#include "Components.h"

#include "../Vector2D.h"

class TransformComponent : public Component

{

public:

Vector2D position;

Vector2D velocity;

Vector2D acceleration;

int speed;

int height = 32;

int width = 32;

int scale = 1;

TransformComponent()

{

position.Zero();

std::cout << "Added TRANFORM COMPONENT\n";

}

TransformComponent(int sc)

{

position.Zero();

scale = sc;

}

TransformComponent(float x, float y)

{

position.x = x;

position.y = y;

}

TransformComponent(float x, float y, int h, int w, int sc)

{

position.x = x;

position.y = y;

height = h;

width = w;

scale = sc;

}

void update() override

{

if (abs(velocity.x) > 0.005f && abs(velocity.y) > 0.005f)

{

position.x += velocity.x;

position.y += velocity.y;

//std::cout << "Position (" << position.x << "; " << position.y << " )\n";

//std::cout << "VELOCITY (" << velocity.x << "; " << velocity.y << " )\n";

velocity.x += acceleration.x;

velocity.y += acceleration.y;

}

else

{

// Устанавливаем скорость и ускорение в 0, если скорость слишком мала

velocity.Zero();

acceleration.Zero();

}

}

void setPos(int x, int y)

{

position.x = x;

position.y = y;

}

void init() override

{

velocity.Zero();

}

};

**Модули, взаимодействующие с ECS:**

**Сollision.cpp:**

#include "Collision.h"

#include "SDL.h"

#include "ECS\ColiderComponent.h"

bool Collision::AABB(const SDL\_Rect& recA, const SDL\_Rect& recB)

{

if (

recA.x + recA.w > recB.x &&

recB.x + recB.w > recA.x &&

recA.y + recA.h > recB.y &&

recB.y + recB.h > recA.y

)

{

return true;

}

return false;

}

bool Collision::AABB(const ColliderComponent& colA, const ColliderComponent& colB)

{

if (AABB(colA.collider, colB.collider))

{

std::cout << colA.tag << " hit: " << colB.tag << std::endl;

return true;

}

else

{

return false;

}

}

**Collision.h:**

#include "Collision.h"

#include "SDL.h"

#include "ECS\ColiderComponent.h"

bool Collision::AABB(const SDL\_Rect& recA, const SDL\_Rect& recB)

{

if (

recA.x + recA.w > recB.x &&

recB.x + recB.w > recA.x &&

recA.y + recA.h > recB.y &&

recB.y + recB.h > recA.y

)

{

return true;

}

return false;

}

bool Collision::AABB(const ColliderComponent& colA, const ColliderComponent& colB)

{

if (AABB(colA.collider, colB.collider))

{

std::cout << colA.tag << " hit: " << colB.tag << std::endl;

return true;

}

else

{

return false;

}

}

**Game.cpp:**

#include "Game.h"

#include "TextureManager.h"

#include "Map.h"

#include <sstream>

#include "ECS/Components.h"

#include "Collision.h"

#include "SDL\_ttf.h"

#include "SDL\_mixer.h"

#include "RecordManager.h"

Entity\* ball;

SDL\_Renderer\* Game::renderer = nullptr;

Map\* map;

RecordManager recordManager;

SDL\_Event Game::event;

bool Game::win = false; // Инициализация переменной win

int currentLevel = 1;

Mix\_Chunk\* holeSound = nullptr;

Mix\_Chunk\* budaSound = nullptr;

float startTime; // Время начала таймера

float elapsedTime; // Прошедшее время в секундах

const char\* mapfile = "assets/2tileset.png";

Game::Game()

:

tiles(manager.getGroup(groupMap)),

balls(manager.getGroup(groupBall)),

holes(manager.getGroup(groupHole)),

borders(manager.getGroup(groupBorder)),

boosters(manager.getGroup(groupBooster)),

flagss(manager.getGroup(groupFlag)),

colliders(manager.getGroup(groupColliders)),

walls(manager.getGroup(groupBorder)),

newPlayer(manager.addEntity()),

hole(manager.addEntity()),

hole1(manager.addEntity()),

flag(manager.addEntity()),

flag1(manager.addEntity()),

wall(manager.addEntity()),

box(manager.addEntity())

{

}

Game::~Game()

{

}

void Game::init(const char \*title, int x, int y, int width, int height, bool fullscreen)

{

int flags = 0;

if (fullscreen)

{

flags = SDL\_WINDOW\_FULLSCREEN;

}

if (SDL\_Init(SDL\_INIT\_EVERYTHING) == 0)

{

std::cout << "Subsytems Initiallized" << std::endl;

window = SDL\_CreateWindow(title, x, y, width, height, flags);

if (window)

{

std::cout << "Window created!" << std::endl;

}

renderer = SDL\_CreateRenderer(window, -1, 0);

if (renderer)

{

SDL\_SetRenderDrawColor(renderer, 255, 255, 255, 255);

std::cout

<< "Renderer created!" << std::endl;

}

isRunning = true;

if (Mix\_OpenAudio(44100, MIX\_DEFAULT\_FORMAT, 2, 2048) < 0) {

std::cerr << "SDL\_mixer could not initialize! SDL\_mixer Error: " << Mix\_GetError() << std::endl;

}

holeSound = Mix\_LoadWAV("assets/sfx/hole.mp3");

if (!holeSound) {

std::cerr << "Failed to load hole sound! SDL\_mixer Error: " << Mix\_GetError() << std::endl;

}

budaSound = Mix\_LoadWAV("assets/sfx/bude.mp3");

if (!budaSound) {

std::cerr << "Failed to load buda sound! SDL\_mixer Error: " << Mix\_GetError() << std::endl;

}

if (budaSound) {

Mix\_PlayChannel(-1, budaSound, 10);

}

}

else

{

isRunning = false;

}

if (TTF\_Init() == -1) {

std::cerr << "SDL\_ttf initialization failed: " << TTF\_GetError() << std::endl;

isRunning = false;

}

map = new Map(manager, "assets/2tileset.png", 1, 32);

startTime = SDL\_GetTicks(); // Записываем время начала в миллисекундах

elapsedTime = 0; // Инициализируем прошедшее время

// ecs implementation

map->LoadMap("assets/map1.map", 30, 19);

newPlayer.addComponent<TransformComponent>(85.0f, 100.0f, 32, 32, 1);

newPlayer.addComponent<SpriteComponent>("assets/ball.png");

newPlayer.addComponent<BallMechanic>();

newPlayer.addComponent<ColliderComponent>("ball");

newPlayer.addGroup(groupBall);

// верхняя стенка

wall.addComponent<TransformComponent>(32.0f, 0.0f, 32, 896, 1);

wall.addComponent<SpriteComponent>("assets/borderup.png");

wall.addComponent<ColliderComponent>("wall");

wall.addGroup(groupColliders);

// стенка слева

auto& wall2(manager.addEntity());

wall2.addComponent<TransformComponent>(0.0f, 32.0f, 566, 32, 1);

wall2.addComponent<SpriteComponent>("assets/borderleft.png");

wall2.addComponent<ColliderComponent>("wall");

wall2.addGroup(groupColliders);

// стенка справа

auto& wall3(manager.addEntity());

wall3.addComponent<TransformComponent>(928.0f, 32.0f, 566, 32, 1);

wall3.addComponent<SpriteComponent>("assets/borderright.png");

wall3.addComponent<ColliderComponent>("wall");

wall3.addGroup(groupColliders);

// стенка снизу

auto& wall4(manager.addEntity());

wall4.addComponent<TransformComponent>(32.0f, 598.0f, 32, 896, 1);

wall4.addComponent<SpriteComponent>("assets/borderdown.png");

wall4.addComponent<ColliderComponent>("wall");

wall4.addGroup(groupColliders);

// 4 квадратика по краям

auto& wall5(manager.addEntity());

wall5.addComponent<TransformComponent>(0.0f, 0.0f, 32, 32, 1);

wall5.addComponent<SpriteComponent>("assets/border.png");

wall5.addComponent<ColliderComponent>("wall");

wall5.addGroup(groupColliders);

auto& wall6(manager.addEntity());

wall6.addComponent<TransformComponent>(928.0f, 598.0f, 32, 32, 1);

wall6.addComponent<SpriteComponent>("assets/border.png");

wall6.addComponent<ColliderComponent>("wall");

wall6.addGroup(groupColliders);

auto& wall7(manager.addEntity());

wall7.addComponent<TransformComponent>(928.0f, 0.0f, 32, 32, 1);

wall7.addComponent<SpriteComponent>("assets/border.png");

wall7.addComponent<ColliderComponent>("wall");

wall7.addGroup(groupColliders);

auto& wall8(manager.addEntity());

wall8.addComponent<TransformComponent>(0.0f, 598.0f, 32, 32, 1);

wall8.addComponent<SpriteComponent>("assets/border.png");

wall8.addComponent<ColliderComponent>("wall");

wall8.addGroup(groupColliders);

//

hole.addComponent<TransformComponent>(850.0f, 80.0f, 40, 40, 1);

hole.addComponent<SpriteComponent>("assets/hole.png");

hole.addComponent<ColliderComponent>("hole");

hole.addGroup(groupColliders);

flag.addComponent<TransformComponent>(852.0f, 30.0f, 100, 50, 1);

flag.addComponent<SpriteComponent>("assets/flag.png");

flag.addGroup(groupFlag);

font = TTF\_OpenFont("assets/font/font.ttf", 30);

if (!font) {

std::cerr << "Failed to load font: " << TTF\_GetError() << std::endl;

isRunning = false;

return;

}

}

bool mouseDown = false;

bool mousePressed = false;

void Game::handleEvents()

{

SDL\_PollEvent(&event);

switch (event.type)

{

case SDL\_QUIT:

isRunning = false;

break;

case SDL\_MOUSEBUTTONDOWN :

if (event.button.button == SDL\_BUTTON\_LEFT)

{

mouseDown = true;

mousePressed = true;

}

break;

case SDL\_MOUSEBUTTONUP:

if (event.button.button == SDL\_BUTTON\_LEFT)

{

mouseDown = false;

}

break;

case SDL\_KEYDOWN:

if (event.key.keysym.sym == SDLK\_ESCAPE)

{

isRunning = false;

}

break;

default:

break;

}

}

bool Game::getMousePressed() {

return mousePressed;

}

bool Game::getMouseDown() {

return mouseDown;

}

void Game::update()

{

if (newPlayer.getComponent<BallMechanic>().strokes == 17) {

newLevelStart();

}

SDL\_Rect ballCol = newPlayer.getComponent<ColliderComponent>().collider;

Vector2D ballPos = newPlayer.getComponent<TransformComponent>().position;

manager.refresh();

manager.update();

bool collisionProcessed = false;

for (auto c : colliders)

{

if (collisionProcessed) break;

SDL\_Rect cCol = c->getComponent<ColliderComponent>().collider;

std::string tag = c->getComponent<ColliderComponent>().tag;

if (Collision::AABB(newPlayer.getComponent<ColliderComponent>(), c->getComponent<ColliderComponent>()))

{

collisionProcessed = true;

// Получаем коллайдеры

SDL\_Rect\* ball = &newPlayer.getComponent<ColliderComponent>().collider;

SDL\_Rect\* entity = &c->getComponent<ColliderComponent>().collider;

// Определяем, с какой стороны произошло столкновение

int ballCenterX = ball->x + ball->w / 2;

int ballCenterY = ball->y + ball->h / 2;

int entityCenterX = entity->x + entity->w / 2;

int entityCenterY = entity->y + entity->h / 2;

int deltaX = ballCenterX - entityCenterX;

int deltaY = ballCenterY - entityCenterY;

int intersectX = (ball->w / 2 + entity->w / 2) - abs(deltaX);

int intersectY = (ball->h / 2 + entity->h / 2) - abs(deltaY);

if (ball->x < entity->x + entity->w &&

ball->x + ball->w > entity->x &&

ball->y < entity->y + entity->h &&

ball->y + ball->h > entity->y &&

tag != "hole" && tag != "sand" && tag != "dirt"

&& tag != "boosterright" && tag != "ice" && tag != "UPPDOWNER"

&& tag != "LEFTRIGHTER" && tag != "corUPLEFT"

&& tag != "corUPRIGHT" && tag != "corDOWNLEFT"

&& tag != "corDOWNRIGHT")

{

// Если пересечение по X больше, чем по Y, то столкновение произошло сверху или снизу

if (intersectX > intersectY) {

if (deltaY > 0) {

std::cout << "COLLISION FROM TOP\n";

newPlayer.getComponent<TransformComponent>().position.y += intersectY; // Смещение вверх

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

else {

std::cout << "COLLISION FROM BOTTOM\n";

newPlayer.getComponent<TransformComponent>().position.y -= intersectY; // Смещение вниз

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

}

else {

if (deltaX > 0) {

std::cout << "COLLISION FROM LEFT\n";

newPlayer.getComponent<TransformComponent>().position.x += intersectX; // Смещение вправо

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

else {

std::cout << "COLLISION FROM RIGHT\n";

newPlayer.getComponent<TransformComponent>().position.x -= intersectX; // Смещение влево

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

}

}

else if (tag == "dirt")

{

newPlayer.getComponent<TransformComponent>().velocity.x \*= 0.98;

newPlayer.getComponent<TransformComponent>().velocity.y \*= 0.98;

}

else if (tag == "sand")

{

newPlayer.getComponent<TransformComponent>().velocity.x \*= 0.95;

newPlayer.getComponent<TransformComponent>().velocity.y \*= 0.95;

}

else if (tag == "ice")

{

}

else if (ball->x < entity->x + entity->w &&

ball->x + ball->w > entity->x &&

ball->y < entity->y + entity->h &&

ball->y + ball->h > entity->y && (tag == "UPPDOWNER"))

{

// Если пересечение по X больше, чем по Y, то столкновение произошло сверху или снизу

if (deltaY > 0) {

std::cout << "COLLISION FROM TOP\n";

newPlayer.getComponent<TransformComponent>().position.y += intersectY; // Смещение вверх

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

else {

std::cout << "COLLISION FROM BOTTOM\n";

newPlayer.getComponent<TransformComponent>().position.y -= intersectY; // Смещение вниз

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

}

else if (ball->x < entity->x + entity->w &&

ball->x + ball->w > entity->x &&

ball->y < entity->y + entity->h &&

ball->y + ball->h > entity->y && (tag == "LEFTRIGHTER"))

{

// Если пересечение по X больше, чем по Y, то столкновение произошло сверху или снизу

if (deltaX > 0) {

std::cout << "COLLISION FROM LEFT\n";

newPlayer.getComponent<TransformComponent>().position.x += intersectX; // Смещение вправо

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

else {

std::cout << "COLLISION FROM RIGHT\n";

newPlayer.getComponent<TransformComponent>().position.x -= intersectX; // Смещение влево

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

}

else if (ball->x < entity->x + entity->w &&

ball->x + ball->w > entity->x &&

ball->y < entity->y + entity->h &&

ball->y + ball->h > entity->y && (tag == "corDOWNRIGHT"))

{

// Если пересечение по X больше, чем по Y, то столкновение произошло сверху или снизу

if (intersectX > intersectY) {

if (deltaY > 0) {

std::cout << "COLLISION FROM TOP\n";

newPlayer.getComponent<TransformComponent>().position.y += intersectY; // Смещение вверх

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

}

else {

if (deltaX > 0) {

std::cout << "COLLISION FROM LEFT\n";

newPlayer.getComponent<TransformComponent>().position.x += intersectX; // Смещение вправо

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

}

}

else if (ball->x < entity->x + entity->w &&

ball->x + ball->w > entity->x &&

ball->y < entity->y + entity->h &&

ball->y + ball->h > entity->y && (tag == "corDOWNLEFT"))

{

// Если пересечение по X больше, чем по Y, то столкновение произошло сверху или снизу

if (intersectX > intersectY) {

if (deltaY > 0) {

std::cout << "COLLISION FROM TOP\n";

newPlayer.getComponent<TransformComponent>().position.y += intersectY; // Смещение вверх

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

}

else {

std::cout << "COLLISION FROM RIGHT\n";

newPlayer.getComponent<TransformComponent>().position.x -= intersectX; // Смещение влево

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

}

else if (ball->x < entity->x + entity->w &&

ball->x + ball->w > entity->x &&

ball->y < entity->y + entity->h &&

ball->y + ball->h > entity->y && (tag == "corUPRIGHT"))

{

// Если пересечение по X больше, чем по Y, то столкновение произошло сверху или снизу

if (intersectX > intersectY) {

std::cout << "COLLISION FROM BOTTOM\n";

newPlayer.getComponent<TransformComponent>().position.y -= intersectY; // Смещение вниз

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

else {

if (deltaX > 0) {

std::cout << "COLLISION FROM LEFT\n";

newPlayer.getComponent<TransformComponent>().position.x += intersectX; // Смещение вправо

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

}

}

else if (ball->x < entity->x + entity->w &&

ball->x + ball->w > entity->x &&

ball->y < entity->y + entity->h &&

ball->y + ball->h > entity->y && (tag == "corUPLEFT"))

{

// Если пересечение по X больше, чем по Y, то столкновение произошло сверху или снизу

if (intersectX > intersectY) {

std::cout << "COLLISION FROM BOTTOM\n";

newPlayer.getComponent<TransformComponent>().position.y -= intersectY; // Смещение вниз

newPlayer.getComponent<TransformComponent>().velocity.y \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.y \*= -1;

}

else {

std::cout << "COLLISION FROM RIGHT\n";

newPlayer.getComponent<TransformComponent>().position.x -= intersectX; // Смещение влево

newPlayer.getComponent<TransformComponent>().velocity.x \*= -1;

newPlayer.getComponent<TransformComponent>().acceleration.x \*= -1;

}

}

else if (tag == "hole")

{

if (holeSound) {

Mix\_PlayChannel(-1, holeSound, 0);

}

if (newPlayer.getComponent<BallMechanic>().strokes < recordManager.getRecord(currentLevel)) {

recordManager.setRecord(currentLevel, newPlayer.getComponent<BallMechanic>().strokes); // Устанавливаем новый рекорд

}

newLevelStart();

}

}

}

elapsedTime = (SDL\_GetTicks64() - startTime) / 1000.0f; // Конвертируем в секунды

}

void Game::loadLevel(const char\* mapPath, int playerPositionX, int playerPositionY, int holePositionX, int holePositionY,

int flagPositionX, int flagPositionY)

{

map->LoadMap(mapPath, 30, 19);

newPlayer.getComponent<TransformComponent>().velocity.x = 0;

newPlayer.getComponent<TransformComponent>().velocity.y = 0;

newPlayer.getComponent<TransformComponent>().position.x = playerPositionX;

newPlayer.getComponent<TransformComponent>().position.y = playerPositionY;

hole.getComponent<TransformComponent>().position.x = holePositionX;

hole.getComponent<TransformComponent>().position.y = holePositionY;

flag.getComponent<TransformComponent>().position.x = flagPositionX;

flag.getComponent<TransformComponent>().position.y = flagPositionY;

}

void Game::newLevelStart()

{

startTime = SDL\_GetTicks();

newPlayer.getComponent<BallMechanic>().strokes = 0;

for (auto& tile : tiles)

{

tile->destroy();

}

for (auto& col : colliders)

{

SDL\_Rect coldestroy = col->getComponent<ColliderComponent>().collider;

std::string tagdestroy = col->getComponent<ColliderComponent>().tag;

if (tagdestroy != "wall" && tagdestroy != "hole")

col->destroy();

}

currentLevel++;

if (currentLevel == 2)

{

loadLevel("assets/map2.map", 50.0f, 550.0f, 870.0f, 500.0f, 872.0f, 450.0f);

}

if (currentLevel == 3)

{

map->LoadMap("assets/map3.map", 30, 19);

loadLevel("assets/map3.map", 50.0f, 300.0f, 770.0f, 550.0f, 772.0f, 500.0f);

hole1.addComponent<TransformComponent>(770.0f, 50.0f, 40, 40, 1);

hole1.addComponent<SpriteComponent>("assets/hole.png");

hole1.addComponent<ColliderComponent>("hole");

hole1.addGroup(groupColliders);

flag1.addComponent<TransformComponent>(772.0f, 0.0f, 100, 50, 1);

flag1.addComponent<SpriteComponent>("assets/flag.png");

flag1.addGroup(groupFlag);

}

if (currentLevel == 4)

{

hole1.destroy();

flag1.destroy();

map->LoadMap("assets/map4.map", 30, 19);

loadLevel("assets/map4.map", 130.0f, 540.0f, 800.0f, 550.0f, 802.0f, 500.0f);

}

if (currentLevel == 5)

{

map->LoadMap("assets/map5.map", 30, 19);

loadLevel("assets/map5.map", 130.0f, 540.0f, 800.0f, 550.0f, 802.0f, 500.0f);

}

if (currentLevel == 6)

{

map->LoadMap("assets/map6.map", 30, 19);

loadLevel("assets/map6.map", 130.0f, 540.0f, 800.0f, 550.0f, 802.0f, 500.0f);

}

if (currentLevel == 7)

{

map->LoadMap("assets/map7.map", 30, 19);

loadLevel("assets/map7.map", 90.0f, 540.0f, 830.0f, 50.0f, 832.0f, 0.0f);

}

if (currentLevel == 8)

{

map->LoadMap("assets/map8.map", 30, 19);

loadLevel("assets/map8.map", 90.0f, 540.0f, 830.0f, 50.0f, 832.0f, 0.0f);

}

if (currentLevel == 9)

{

map->LoadMap("assets/map9.map", 30, 19);

loadLevel("assets/map9.map", 90.0f, 540.0f, 830.0f, 50.0f, 832.0f, 0.0f);

}

}

void Game::render()

{

SDL\_RenderClear(renderer);

for (auto& tile : tiles)

{

tile->draw();

}

for (auto& border : borders)

{

border->draw();

}

for (auto& c : colliders)

{

c->draw();

}

for (auto& hole : holes)

{

hole->draw();

}

for (auto& booster : boosters)

{

booster->draw();

}

for (auto& ball : balls)

{

ball->draw();

}

for (auto& flag : flagss)

{

flag->draw();

}

std::string strokesText = "Strokes: " + std::to\_string(newPlayer.getComponent<BallMechanic>().strokes);

renderText(renderer, font, strokesText, 33, 0); // Текст в левом верхнем углу

if (newPlayer.getComponent<BallMechanic>().strokes == 16) {

std::string strokesLimitYesText = "No more strokes, hit to continue";

renderText(renderer, font, strokesLimitYesText, 390, 600);

}

std::string strokesLimitText = "Limit - 16";

renderText(renderer, font, strokesLimitText, 230, 0);

std::string levelText = "Level: " + std::to\_string(currentLevel);

renderText(renderer, font, levelText, 900 - 60, 0);

std::string nameText = "CursGolf";

renderText(renderer, font, nameText, 900 - 480, 0);

std::string escText = "esc for exit";

renderText(renderer, font, escText, 32, 596);

//recordManager.loadRecords();

std::string recordText = "Record: " + std::to\_string(recordManager.getRecord(currentLevel));

renderText(renderer, font, recordText, 820, 596); // Позиция в правом нижнем углу

std::ostringstream timerStream;

timerStream.precision(1);

timerStream << std::fixed << elapsedTime;

std::string timerText = "Time: " + timerStream.str() + "s";

renderText(renderer, font, timerText, 620, 0);

SDL\_RenderPresent(renderer);

}

void Game::clean()

{

currentLevel = 1;

SDL\_DestroyWindow(window);

SDL\_DestroyRenderer(renderer);

Mix\_CloseAudio();

SDL\_Quit();

if (font) {

TTF\_CloseFont(font);

}

std::cout << "You've exited the game" << std::endl;

}

void Game::renderText(SDL\_Renderer\* renderer, TTF\_Font\* font, const std::string& text, int x, int y)

{

SDL\_Color textColor = { 0, 0, 0, 255 }; // Черный цвет текста

// Создаём поверхность с текстом

SDL\_Surface\* surface = TTF\_RenderText\_Solid(font, text.c\_str(), textColor);

if (!surface) {

std::cerr << "Failed to create text surface: " << TTF\_GetError() << std::endl;

return;

}

// Создаём текстуру из поверхности

SDL\_Texture\* texture = SDL\_CreateTextureFromSurface(renderer, surface);

if (!texture) {

std::cerr << "Failed to create texture: " << SDL\_GetError() << std::endl;

SDL\_FreeSurface(surface);

return;

}

// Устанавливаем размер текста

SDL\_Rect destRect = { x, y, surface->w, surface->h };

// Освобождаем поверхность

SDL\_FreeSurface(surface);

// Отображаем текстуру

SDL\_RenderCopy(renderer, texture, nullptr, &destRect);

// Удаляем текстуру после отрисовки

SDL\_DestroyTexture(texture);

}

**Game.h:**

#pragma once

#include "SDL.h"

#include "SDL\_ttf.h"

#include "SDL\_image.h"

#include <iostream>

#include <vector>

#include "TextureManager.h"

#include "ECS/ECS.h"

#include "RecordManager.cpp"

class ColliderComponent;

class Game

{

public:

Game();

~Game();

void init(const char \*title, int x, int y, int width, int height, bool fullscreen);

void handleEvents();

bool getMousePressed();

bool getMouseDown();

void update();

void render();

void clean();

bool running() { return isRunning; }

static SDL\_Renderer\* renderer;

static SDL\_Event event;

static bool win; // Добавляем переменную win как статическую

void renderText(SDL\_Renderer\* renderer, TTF\_Font\* font, const std::string& text, int x, int y);

void loadLevel(const char\* mapPath, int playerPositionX, int playerPositionY, int holePositionX, int holePositionY,

int flagPositionX, int flagPositionY);

void newLevelStart();

enum groupLabels : std::size\_t

{

groupMap,

groupBall,

groupColliders,

groupHole,

groupBorder,

groupBooster,

groupFlag

};

Manager manager;

Entity& newPlayer;

Entity& hole;

Entity& hole1;

Entity& flag;

Entity& flag1;

Entity& wall;

Entity& box;

std::vector<Entity\*, std::allocator<Entity\*>>& tiles;

std::vector<Entity\*, std::allocator<Entity\*>>& balls;

std::vector<Entity\*, std::allocator<Entity\*>>& holes;

std::vector<Entity\*, std::allocator<Entity\*>>& borders;

std::vector<Entity\*, std::allocator<Entity\*>>& boosters;

std::vector<Entity\*, std::allocator<Entity\*>>& flagss;

std::vector<Entity\*, std::allocator<Entity\*>>& colliders;

std::vector<Entity\*, std::allocator<Entity\*>>& walls;

private:

int counter = 0;

bool isRunning;

SDL\_Window \*window;

std::vector<std::string> musicTracks;

int currentTrackIndex = 0;

bool isMusicPlaying = false;

TTF\_Font\* font = nullptr;

};

**InfoWindow.cpp:**

#include "InfoWindow.h"

#include <iostream>

#include <vector>

#include <sstream>

InfoWindow::InfoWindow() : running(false) {}

InfoWindow::~InfoWindow() {

SDL\_DestroyWindow(infoWindow);

SDL\_DestroyRenderer(infoRenderer);

TTF\_CloseFont(font);

TTF\_Quit();

SDL\_Quit();

}

void InfoWindow::init(const char\* title, int xpos, int ypos, int width, int height, bool fullscreen) {

int flags = 0;

if (fullscreen) {

flags = SDL\_WINDOW\_FULLSCREEN;

}

if (SDL\_Init(SDL\_INIT\_EVERYTHING) == 0) {

TTF\_Init();

infoWindow = SDL\_CreateWindow(title, xpos, ypos, width, height, flags);

infoRenderer = SDL\_CreateRenderer(infoWindow, -1, 0);

if (infoRenderer) {

SDL\_SetRenderDrawColor(infoRenderer, 255, 255, 255, 255);

}

font = TTF\_OpenFont("assets/font/font.ttf", 20);

if (!font) {

std::cerr << "Не удалось загрузить шрифт: " << TTF\_GetError() << std::endl;

}

running = true;

}

else {

running = false;

}

}

void InfoWindow::handleEvents() {

SDL\_Event event;

while (SDL\_PollEvent(&event)) {

if (event.type == SDL\_QUIT ||

(event.type == SDL\_KEYDOWN && event.key.keysym.sym == SDLK\_RETURN) ||

(event.type == SDL\_KEYDOWN && event.key.keysym.sym == SDLK\_ESCAPE)) {

running = false;

}

}

}

void InfoWindow::render() {

SDL\_RenderClear(infoRenderer);

SDL\_Color textColor = { 0, 0, 0 };

// Текст для отображения

const char\* infoText = "Welcome to the CursGolf!\n- Use arrow keys to navigate\n- Control the ball with your mouse\n- Try to make fewer strokes\n- Try to complete the level in less time\n- Use enter to exit\nEnjoy!\n----------------------------------------\n made by Andrey and Stas from PI-32";

std::vector<std::string> lines;

std::stringstream textStream(infoText);

std::string line;

// Разбиваем текст на строки по символу '\n'

while (std::getline(textStream, line, '\n')) {

lines.push\_back(line);

}

int yOffset = 20; // Начальная позиция по Y

for (const auto& str : lines) {

SDL\_Surface\* textSurface = TTF\_RenderText\_Solid(font, str.c\_str(), textColor);

SDL\_Texture\* textTexture = SDL\_CreateTextureFromSurface(infoRenderer, textSurface);

SDL\_Rect textRect = { 20, yOffset, textSurface->w, textSurface->h };

SDL\_FreeSurface(textSurface);

SDL\_RenderCopy(infoRenderer, textTexture, NULL, &textRect);

SDL\_DestroyTexture(textTexture);

yOffset += 30; // Смещение по Y для следующей строки

}

SDL\_RenderPresent(infoRenderer);

}

bool InfoWindow::isRunning() const {

return running;

}

**InfoWindow.h:**

#ifndef INFOWINDOW\_H

#define INFOWINDOW\_H

#include <SDL.h>

#include <SDL\_ttf.h>

class InfoWindow {

public:

InfoWindow();

~InfoWindow();

void init(const char\* title, int xpos, int ypos, int width, int height, bool fullscreen);

void handleEvents();

void render();

bool isRunning() const;

private:

bool running;

SDL\_Window\* infoWindow;

SDL\_Renderer\* infoRenderer;

TTF\_Font\* font;

};

#endif

**Main.cpp:**

#include "Game.h"

#include "menu.h"

#include "InfoWindow.h"

#include "RecordManager.h"

#include "RecordWindow.h"

using namespace std;

const int SCREEN\_WIDTH = 960;

const int SCREEN\_HEIGHT = 630;

SDL\_Window \*window = nullptr;

SDL\_Surface \*screen = NULL;

SDL\_Renderer \*renderer = nullptr;

Game \*game = nullptr;

Menu\* menu = nullptr;

InfoWindow\* infoWindow = nullptr;

RecordWindow\* recordWindow = nullptr;

int main(int, char \*\*)

{

const int FPS = 144;

const int frameDelay = 1000 / FPS;

Uint32 frameStart;

int frameTime;

bool gameRunning = true;

while (gameRunning) {

menu = new Menu();

menu->init("CursGolf", SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED, 600, 350, false);

while (menu->isRunning()) {

menu->handleEvents();

menu->render();

}

int selectedOption = menu->getSelectedOption();

delete menu; // Уничтожаем меню перед началом игры

if (selectedOption == 0) { // Если выбрана "Игра"

game = new Game();

game->init("CursGolf", SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED, SCREEN\_WIDTH, SCREEN\_HEIGHT, false);

while (game->running()) {

frameStart = SDL\_GetTicks();

game->handleEvents();

game->update();

game->render();

frameTime = SDL\_GetTicks() - frameStart;

if (frameDelay > frameTime) {

SDL\_Delay(frameDelay - frameTime);

}

}

game->clean();

delete game;

}

else if (selectedOption == 1)

{

recordWindow = new RecordWindow();

recordWindow->init("Game Records", SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED, 400, 400, false);

while (recordWindow->isRunning()) {

recordWindow->handleEvents();

recordWindow->render();

}

delete recordWindow;

}

else if (selectedOption == 2) { // Если выбрана "Info"

infoWindow = new InfoWindow();

infoWindow->init("Game Info", SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED, 400, 300, false);

while (infoWindow->isRunning()) {

infoWindow->handleEvents();

infoWindow->render();

}

delete infoWindow;

}

else if (selectedOption == -1 || selectedOption == 3) { // Если выбран "Выход" или окно закрыто

gameRunning = false;

}

}

return 0;

}

**Map.cpp:**

#include "Map.h"

#include <fstream>

#include "ECS/ECS.h"

#include "ECS/Components.h"

extern Manager manager;

Map::Map(Manager& manager, const char\* mfp, int ms, int ts) : manager(manager), mapFilePath(mfp), mapScale(ms), tileSize(ts)

{

scaledSize = ms \* ts;

}

// Загружаем карту

void Map::LoadMap(std::string path, int sizeX, int sizeY)

{

char tile;

std::fstream mapFile;

mapFile.open(path); // открываем файл с картой

int srcX, srcY;

for (int y = 0; y < sizeY; y++)

{

for (int x = 0; x < sizeX; x++)

{

mapFile.get(tile);

srcY = atoi(&tile) \* tileSize;

mapFile.get(tile);

srcX = atoi(&tile) \* tileSize;

std::cout << tile;

AddTile(srcX, srcY, x \* scaledSize, y \* scaledSize

);

mapFile.ignore();

}

std::cout << "\n";

}

mapFile.ignore();

std::cout << "\n";

std::cout << "\n";

// генерируем коллизию

for (int y = 0; y < sizeY; y++)

{

for (int x = 0; x < sizeX; x++)

{

mapFile.get(tile);

std::cout << tile;

if (tile == '1') {

Entity& tcol(manager.addEntity());

tcol.addComponent<ColliderComponent>("terrain", x\*scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol.addGroup(Game::groupColliders);

}

if (tile == '2') {

Entity& tcol2(manager.addEntity());

tcol2.addComponent<ColliderComponent>("UPPDOWNER", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol2.addGroup(Game::groupColliders);

}

if (tile == '3') {

Entity& tcol3(manager.addEntity());

tcol3.addComponent<ColliderComponent>("LEFTRIGHTER", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol3.addGroup(Game::groupColliders);

}

if (tile == '4') {

Entity& tcol4(manager.addEntity());

tcol4.addComponent<ColliderComponent>("corUPLEFT", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol4.addGroup(Game::groupColliders);

}

if (tile == '5') {

auto& tcol5(manager.addEntity());

tcol5.addComponent<ColliderComponent>("corUPRIGHT", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol5.addGroup(Game::groupColliders);

}

if (tile == '6') {

auto& tcol6(manager.addEntity());

tcol6.addComponent<ColliderComponent>("corDOWNLEFT", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol6.addGroup(Game::groupColliders);

}

if (tile == '7') {

auto& tcol7(manager.addEntity());

tcol7.addComponent<ColliderComponent>("corDOWNRIGHT", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol7.addGroup(Game::groupColliders);

}

if (tile == '8') {

auto& tcol8(manager.addEntity());

tcol8.addComponent<ColliderComponent>("dirt", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol8.addGroup(Game::groupColliders);

}

if (tile == '9') {

auto& tcol9(manager.addEntity());

tcol9.addComponent<ColliderComponent>("sand", x \* scaledSize, y \* scaledSize, tileSize \* mapScale);

tcol9.addGroup(Game::groupColliders);

}

mapFile.ignore();

}

std::cout << "\n";

}

std::cout << "Map generated\n";

mapFile.close();

}

void Map::AddTile(int srcX, int srcY, int xpos, int ypos)

{

Entity& tile(manager.addEntity());

tile.addComponent<TileComponent>(srcX, srcY, xpos, ypos,tileSize,mapScale, mapFilePath);

tile.addGroup(Game::groupMap);

}

**Map.h:**

#pragma once

#include <string>

#include "Game.h"

class Map {

public:

Map(Manager& manager, const char\* mfp, int ms, int ts);

~Map();

void LoadMap(std::string path, int sizeX, int sizeY);

void AddTile(int srcX, int srcY, int xpos, int ypos);

private:

Manager& manager;

const char\* mapFilePath;

int mapScale;

int tileSize;

int scaledSize;

};

**Menu.cpp:**

#include "Menu.h"

#include <iostream>

Menu::Menu() : running(false), selectedOption(0) {}

Menu::~Menu() {

SDL\_DestroyWindow(menuWindow);

SDL\_DestroyRenderer(menuRenderer);

TTF\_CloseFont(font);

TTF\_Quit();

SDL\_Quit();

}

void Menu::init(const char\* title, int xpos, int ypos, int width, int height, bool fullscreen) {

int flags = 0;

if (fullscreen) {

flags = SDL\_WINDOW\_FULLSCREEN;

}

if (SDL\_Init(SDL\_INIT\_EVERYTHING) == 0) {

TTF\_Init();

menuWindow = SDL\_CreateWindow(title, xpos, ypos, width, height, flags);

menuRenderer = SDL\_CreateRenderer(menuWindow, -1, 0);

if (menuRenderer) {

SDL\_SetRenderDrawColor(menuRenderer, 200, 255, 200, 255);

}

font = TTF\_OpenFont("assets/font/font.ttf", 30);

if (!font) {

std::cerr << "Не удалось загрузить шрифт: " << TTF\_GetError() << std::endl;

}

running = true;

menuOptions = { "Game", "Records", "Info", "Exit" };

}

else {

running = false;

}

}

void Menu::handleEvents() {

SDL\_Event event;

while (SDL\_PollEvent(&event)) {

switch (event.type) {

case SDL\_QUIT:

selectedOption = -1; // Устанавливаем опцию -1, чтобы отличить закрытие окна

running = false;

break;

case SDL\_KEYDOWN:

switch (event.key.keysym.sym) {

case SDLK\_UP:

selectedOption = (selectedOption > 0) ? selectedOption - 1 : menuOptions.size() - 1;

break;

case SDLK\_DOWN:

selectedOption = (selectedOption < menuOptions.size() - 1) ? selectedOption + 1 : 0;

break;

case SDLK\_RETURN:

running = false;

break;

}

}

}

}

void Menu::render() {

SDL\_RenderClear(menuRenderer);

// Рендерим заголовок "CursGolf"

SDL\_Color whiteColor = { 0, 0, 0 };

SDL\_Surface\* titleSurface = TTF\_RenderText\_Solid(font, "CursGolf", whiteColor);

SDL\_Texture\* titleTexture = SDL\_CreateTextureFromSurface(menuRenderer, titleSurface);

SDL\_Rect titleRect;

titleRect.x = 250;

titleRect.y = 40;

titleRect.w = titleSurface->w;

titleRect.h = titleSurface->h;

SDL\_FreeSurface(titleSurface);

SDL\_RenderCopy(menuRenderer, titleTexture, NULL, &titleRect);

SDL\_DestroyTexture(titleTexture);

for (int i = 0; i < menuOptions.size(); ++i) {

SDL\_Color color = (i == selectedOption) ? SDL\_Color{ 0, 150, 0 } : SDL\_Color{ 0, 0, 0 };

SDL\_Surface\* surfaceMessage = TTF\_RenderText\_Solid(font, menuOptions[i].c\_str(), color);

SDL\_Texture\* message = SDL\_CreateTextureFromSurface(menuRenderer, surfaceMessage);

SDL\_Rect messageRect;

messageRect.x = 100;

messageRect.y = 100 + i \* 50;

messageRect.w = surfaceMessage->w;

messageRect.h = surfaceMessage->h;

SDL\_FreeSurface(surfaceMessage);

SDL\_RenderCopy(menuRenderer, message, NULL, &messageRect);

SDL\_DestroyTexture(message);

}

SDL\_RenderPresent(menuRenderer);

}

**Menu.h:**

#pragma once

#include <SDL.h>

#include <SDL\_ttf.h>

#include <vector>

#include <string>

class Menu {

public:

Menu();

~Menu();

void init(const char\* title, int xpos, int ypos, int width, int height, bool fullscreen);

void handleEvents();

void render();

bool isRunning() { return running; }

int getSelectedOption() { return selectedOption; }

private:

bool running;

int selectedOption;

SDL\_Window\* menuWindow;

SDL\_Renderer\* menuRenderer;

TTF\_Font\* font;

std::vector<std::string> menuOptions;

};

**RecordManager.cpp:**

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <sstream>

#include <limits>

class RecordManager {

private:

std::vector<int> records;

public:

RecordManager() {

// Инициализация рекордов для 10 уровней, начальное значение - бесконечность

records.resize(10, std::numeric\_limits<int>::max());

loadRecords();

}

// Загрузка рекордов из файла

void loadRecords() {

std::ifstream file("assets/records/records.txt");

if (file.is\_open()) {

std::string line;

int level = 0;

while (std::getline(file, line) && level < 10) {

records[level] = std::stoi(line);

level++;

}

file.close();

}

}

// Сохранение рекордов в файл

void saveRecords() {

std::ofstream file("assets/records/records.txt");

if (file.is\_open()) {

for (int i = 0; i < 10; i++) {

file << records[i] << "\n";

}

file.close();

}

}

// Получить рекорд для определенного уровня

int getRecord(int level) const {

if (level >= 1 && level <= 10) {

return records[level - 1];

}

return std::numeric\_limits<int>::max();

}

// Установить новый рекорд для определенного уровня

void setRecord(int level, int strokes) {

if (level >= 1 && level <= 10) {

if (strokes < records[level - 1]) {

records[level - 1] = strokes;

saveRecords(); // Перезаписываем файл с новыми рекордами

}

}

}

};

**RecordManager.h :**

**RecordWindow.cpp:**

#include "RecordWindow.h"

#include <iostream>

#include <fstream>

#include <vector>

#include <sstream>

RecordWindow::RecordWindow() : running(false) {}

RecordWindow::~RecordWindow() {

SDL\_DestroyWindow(recordWindow);

SDL\_DestroyRenderer(recordRenderer);

TTF\_CloseFont(font);

TTF\_Quit();

SDL\_Quit();

}

void RecordWindow::init(const char\* title, int xpos, int ypos, int width, int height, bool fullscreen) {

int flags = 0;

if (fullscreen) {

flags = SDL\_WINDOW\_FULLSCREEN;

}

if (SDL\_Init(SDL\_INIT\_EVERYTHING) == 0) {

TTF\_Init();

recordWindow = SDL\_CreateWindow(title, xpos, ypos, width, height, flags);

recordRenderer = SDL\_CreateRenderer(recordWindow, -1, 0);

if (recordRenderer) {

SDL\_SetRenderDrawColor(recordRenderer, 255, 255, 255, 255);

}

font = TTF\_OpenFont("assets/font/font.ttf", 20);

if (!font) {

std::cerr << "Не удалось загрузить шрифт: " << TTF\_GetError() << std::endl;

}

running = true;

}

else {

running = false;

}

}

void RecordWindow::handleEvents() {

SDL\_Event event;

while (SDL\_PollEvent(&event)) {

if (event.type == SDL\_QUIT ||

(event.type == SDL\_KEYDOWN && event.key.keysym.sym == SDLK\_RETURN) ||

(event.type == SDL\_KEYDOWN && event.key.keysym.sym == SDLK\_ESCAPE)) {

running = false;

}

}

}

void RecordWindow::render() {

SDL\_RenderClear(recordRenderer);

SDL\_Color textColor = { 0, 0, 0 };

// Загружаем рекорды из файла

std::vector<std::string> records = loadRecords("assets/records/records.txt");

// Заголовок

std::string header = "Level Records:";

std::string minus = "----------------------------------------";

std::vector<std::string> lines;

lines.push\_back(header);

lines.push\_back(minus);

// Добавляем рекорды в строки для отображения

for (size\_t i = 0; i < records.size(); ++i) {

std::string record = "Level " + std::to\_string(i + 1) + ": " + records[i] + " strokes";

lines.push\_back(record);

}

// Отображаем текст

int yOffset = 20; // Начальная позиция по Y

for (const auto& str : lines) {

SDL\_Surface\* textSurface = TTF\_RenderText\_Solid(font, str.c\_str(), textColor);

SDL\_Texture\* textTexture = SDL\_CreateTextureFromSurface(recordRenderer, textSurface);

SDL\_Rect textRect = { 20, yOffset, textSurface->w, textSurface->h };

SDL\_FreeSurface(textSurface);

SDL\_RenderCopy(recordRenderer, textTexture, NULL, &textRect);

SDL\_DestroyTexture(textTexture);

yOffset += 30; // Смещение по Y для следующей строки

}

SDL\_RenderPresent(recordRenderer);

}

bool RecordWindow::isRunning() const {

return running;

}

std::vector<std::string> RecordWindow::loadRecords(const std::string& filepath) {

std::vector<std::string> records;

std::ifstream file(filepath);

if (!file) {

std::cerr << "Не удалось открыть файл с рекордами: " << filepath << std::endl;

return records;

}

std::string line;

while (std::getline(file, line)) {

records.push\_back(line);

}

file.close();

return records;

}

**RecordWindow.h :**

#ifndef RECORDWINDOW\_H

#define RECORDWINDOW\_H

#include <SDL.h>

#include <SDL\_ttf.h>

#include <vector>

#include <string>

class RecordWindow {

public:

RecordWindow();

~RecordWindow();

void init(const char\* title, int xpos, int ypos, int width, int height, bool fullscreen);

void handleEvents();

void render();

bool isRunning() const;

private:

SDL\_Window\* recordWindow;

SDL\_Renderer\* recordRenderer;

TTF\_Font\* font;

bool running;

std::vector<std::string> loadRecords(const std::string& filepath);

};

#endif // RECORDWINDOW\_H

**TextureManager.cpp:**

#include "TextureManager.h"

SDL\_Texture\* TextureManager::LoadTexture(const char\* texture)

{

SDL\_Surface\* tempsurf = IMG\_Load(texture);

SDL\_Texture\* tex = SDL\_CreateTextureFromSurface(Game::renderer, tempsurf);

SDL\_FreeSurface(tempsurf);

return tex;

}

void TextureManager::Draw(SDL\_Texture\* texture, SDL\_Rect src, SDL\_Rect dest)

{

SDL\_RenderCopy(Game::renderer, texture, &src, &dest);

}

**TextureManager.h :**

#pragma once

#include "Game.h"

class TextureManager {

public:

static SDL\_Texture\* LoadTexture(const char\* fileName);

static void Draw(SDL\_Texture \* texture, SDL\_Rect src, SDL\_Rect dest);

};

**Vector2D.cpp:**

#include "Vector2D.h"

Vector2D::Vector2D()

{

x = 0.0f;

y = 0.0f;

}

Vector2D::Vector2D(float x, float y)

{

this->x = x;

this->y = y;

}

Vector2D& Vector2D::Add(const Vector2D& vec)

{

this->x += vec.x;

this->y += vec.y;

return \*this;

}

Vector2D& Vector2D::Subtract(const Vector2D& vec)

{

this->x -= vec.x;

this->y -= vec.y;

return \*this;

}

Vector2D& Vector2D::Multiply(const Vector2D& vec)

{

this->x \*= vec.x;

this->y \*= vec.y;

return \*this;

}

Vector2D& Vector2D::Divide(const Vector2D& vec)

{

this->x /= vec.x;

this->y /= vec.y;

return \*this;

}

Vector2D& operator+(Vector2D& v1, const Vector2D& v2)

{

return v1.Add(v2);

}

Vector2D& operator-(Vector2D& v1, const Vector2D& v2)

{

return v1.Subtract(v2);

}

Vector2D& operator\*(Vector2D& v1, const Vector2D& v2)

{

return v1.Multiply(v2);

}

Vector2D& operator/(Vector2D& v1, const Vector2D& v2)

{

return v1.Divide(v2);

}

Vector2D& Vector2D::operator+=(const Vector2D& vec)

{

return this->Add(vec);

}

Vector2D& Vector2D::operator-=(const Vector2D& vec)

{

return this->Subtract(vec);

}

Vector2D& Vector2D::operator\*=(const Vector2D& vec)

{

return this->Multiply(vec);

}

Vector2D& Vector2D::operator/=(const Vector2D& vec)

{

return this->Divide(vec);

}

Vector2D& Vector2D::operator\*(const int& i)

{

this->x \*= i;

this->y \*= i;

return \*this;

}

Vector2D& Vector2D::Zero()

{

this->x = 0;

this->y = 0;

return \*this;

}

std::ostream& operator<<(std::ostream& stream, const Vector2D& vec)

{

stream << "(" << vec.x << "," << vec.y << ")";

return stream;

}

**Vector2D.h:**

#pragma once

#include <iostream>

class Vector2D

{

public:

float x;

float y;

Vector2D();

Vector2D(float x, float y);

Vector2D& Add(const Vector2D& vec);

Vector2D& Subtract(const Vector2D& vec);

Vector2D& Multiply(const Vector2D& vec);

Vector2D& Divide(const Vector2D& vec);

friend Vector2D& operator+(Vector2D& v1, const Vector2D& v2);

friend Vector2D& operator-(Vector2D& v1, const Vector2D& v2);

friend Vector2D& operator\*(Vector2D& v1, const Vector2D& v2);

friend Vector2D& operator/(Vector2D& v1, const Vector2D& v2);

Vector2D& operator+=(const Vector2D& vec);

Vector2D& operator-=(const Vector2D& vec);

Vector2D& operator\*=(const Vector2D& vec);

Vector2D& operator/=(const Vector2D& vec);

Vector2D& operator\*(const int& i);

Vector2D& Zero();

friend std::ostream& operator<<(std::ostream& stream, const Vector2D& vec);

};

**Прототип GolfGame.java на языке java:**

import org.lwjgl.system.\*;  
  
import java.nio.\*;  
  
import static org.lwjgl.glfw.Callbacks.\*;  
import static org.lwjgl.glfw.GLFW.\*;  
import static org.lwjgl.opengl.GL11.\*;  
import static org.lwjgl.system.MemoryStack.\*;  
import static org.lwjgl.system.MemoryUtil.\*;  
  
public class GolfGame {  
  
 private long window;  
 private float ballX = 0.0f;  
 private float ballY = 0.0f;  
 private float ballRadius = 0.05f;  
 private float holeX = 0.5f;  
 private float holeY = 0.5f;  
 private float holeRadius = 0.06f;  
  
 private boolean isDragging = false;  
 private float dragStartX, dragStartY;  
 private float velocityX = 0.0f;  
 private float velocityY = 0.0f;  
  
 private float[][] obstacles = {  
 {-0.5f, 0.0f, 0.2f, 0.3f}, *// Obstacle 1: x, y, width, height* {0.2f, -0.3f, 0.2f, 0.2f}, *// Obstacle 2* {-0.2f, 0.4f, 0.25f, 0.3f} *// Obstacle 3* };  
  
 public void run() {  
 System.*out*.println("Starting Mini Golf Game!");  
  
 init();  
 loop();  
  
 *glfwFreeCallbacks*(window);  
 *glfwDestroyWindow*(window);  
  
 *glfwTerminate*();  
 *glfwSetErrorCallback*(null).free();  
 }  
  
 private void init() {  
 GLFWErrorCallback.*createPrint*(System.*err*).set();  
  
 if (!*glfwInit*())  
 throw new IllegalStateException("Unable to initialize GLFW");  
  
 *glfwDefaultWindowHints*();  
 *glfwWindowHint*(*GLFW\_VISIBLE*, *GLFW\_FALSE*);  
 *glfwWindowHint*(*GLFW\_RESIZABLE*, *GLFW\_TRUE*);  
  
 window = *glfwCreateWindow*(800, 600, "CursGolfJava", *NULL*, *NULL*);  
 if (window == *NULL*)  
 throw new RuntimeException("Failed to create the GLFW window");  
  
 *glfwSetCursorPosCallback*(window, this::onMouseMove);  
 *glfwSetMouseButtonCallback*(window, this::onMouseClick);  
  
 try (MemoryStack stack = *stackPush*()) {  
 IntBuffer pWidth = stack.mallocInt(1);  
 IntBuffer pHeight = stack.mallocInt(1);  
  
 *glfwGetWindowSize*(window, pWidth, pHeight);  
 GLFWVidMode vidmode = *glfwGetVideoMode*(*glfwGetPrimaryMonitor*());  
  
 *glfwSetWindowPos*(  
 window,  
 (vidmode.width() - pWidth.get(0)) / 2,  
 (vidmode.height() - pHeight.get(0)) / 2  
 );  
 }  
  
 *glfwMakeContextCurrent*(window);  
 *glfwSwapInterval*(1);  
  
 *glfwShowWindow*(window);  
 }  
  
 private void loop() {  
 GL.*createCapabilities*();  
 *glClearColor*(0.0f, 0.6f, 0.2f, 0.0f);  
  
 while (!*glfwWindowShouldClose*(window)) {  
 *glClear*(*GL\_COLOR\_BUFFER\_BIT* | *GL\_DEPTH\_BUFFER\_BIT*);  
  
 updateBallPosition();  
 drawHole();  
 drawObstacles();  
 drawBall();  
  
 if (checkWinCondition()) {  
 System.*out*.println("You Win!");  
 break;  
 }  
  
 *glfwSwapBuffers*(window);  
 *glfwPollEvents*();  
 }  
 }  
  
 private void drawBall() {  
 *glColor3f*(1.0f, 1.0f, 1.0f);  
 drawCircle(ballX, ballY, ballRadius);  
 }  
  
 private void drawHole() {  
 *glColor3f*(0.0f, 0.0f, 0.0f);  
 drawCircle(holeX, holeY, holeRadius);  
 }  
  
 private void drawObstacles() {  
 *glColor3f*(0.0f, 0.0f, 0.0f);  
 for (float[] obstacle : obstacles) {  
 drawRectangle(obstacle[0], obstacle[1], obstacle[2], obstacle[3]);  
 }  
 }  
  
 private void drawRectangle(float x, float y, float width, float height) {  
 *glBegin*(*GL\_QUADS*);  
 *glVertex2f*(x - width / 2, y - height / 2);  
 *glVertex2f*(x + width / 2, y - height / 2);  
 *glVertex2f*(x + width / 2, y + height / 2);  
 *glVertex2f*(x - width / 2, y + height / 2);  
 *glEnd*();  
 }  
  
 private void drawCircle(float x, float y, float radius) {  
 *glBegin*(*GL\_TRIANGLE\_FAN*);  
 *glVertex2f*(x, y);  
 for (int i = 0; i <= 360; i++) {  
 double angle = Math.*toRadians*(i);  
 *glVertex2f*((float) (x + Math.*cos*(angle) \* radius), (float) (y + Math.*sin*(angle) \* radius));  
 }  
 *glEnd*();  
 }  
  
 private void onMouseMove(long window, double xpos, double ypos) {}  
  
 private void onMouseClick(long window, int button, int action, int mods) {  
 if (button == *GLFW\_MOUSE\_BUTTON\_LEFT*) {  
 if (action == *GLFW\_PRESS*) {  
 isDragging = true;  
  
 try (MemoryStack stack = *stackPush*()) {  
 DoubleBuffer mouseX = stack.mallocDouble(1);  
 DoubleBuffer mouseY = stack.mallocDouble(1);  
 *glfwGetCursorPos*(window, mouseX, mouseY);  
  
 dragStartX = (float) (mouseX.get(0) / 400.0 - 1.0);  
 dragStartY = (float) -(mouseY.get(0) / 300.0 - 1.0);  
 }  
 } else if (action == *GLFW\_RELEASE*) {  
 isDragging = false;  
  
 float dragEndX;  
 float dragEndY;  
  
 try (MemoryStack stack = *stackPush*()) {  
 DoubleBuffer mouseX = stack.mallocDouble(1);  
 DoubleBuffer mouseY = stack.mallocDouble(1);  
 *glfwGetCursorPos*(window, mouseX, mouseY);  
  
 dragEndX = (float) (mouseX.get(0) / 400.0 - 1.0);  
 dragEndY = (float) -(mouseY.get(0) / 300.0 - 1.0);  
 }  
  
 velocityX = dragStartX - dragEndX;  
 velocityY = dragStartY - dragEndY;  
 }  
 }  
 }  
  
 private void updateBallPosition() {  
 ballX += velocityX \* 0.01f;  
 ballY += velocityY \* 0.01f;  
  
 velocityX \*= 0.98f; *// Friction* velocityY \*= 0.98f;  
  
 if (Math.*abs*(velocityX) < 0.001f) velocityX = 0;  
 if (Math.*abs*(velocityY) < 0.001f) velocityY = 0;  
  
 if (ballX - ballRadius < -1.0f || ballX + ballRadius > 1.0f) {  
 velocityX = -velocityX;  
 ballX = Math.*max*(ballX, -1.0f + ballRadius);  
 ballX = Math.*min*(ballX, 1.0f - ballRadius);  
 }  
 if (ballY - ballRadius < -1.0f || ballY + ballRadius > 1.0f) {  
 velocityY = -velocityY;  
 ballY = Math.*max*(ballY, -1.0f + ballRadius);  
 ballY = Math.*min*(ballY, 1.0f - ballRadius);  
 }  
  
 for (float[] obstacle : obstacles) {  
 float left = obstacle[0] - obstacle[2] / 2;  
 float right = obstacle[0] + obstacle[2] / 2;  
 float bottom = obstacle[1] - obstacle[3] / 2;  
 float top = obstacle[1] + obstacle[3] / 2;  
  
 if (ballX + ballRadius > left && ballX - ballRadius < right &&  
 ballY + ballRadius > bottom && ballY - ballRadius < top) {  
 if (ballX < left || ballX > right) velocityX = -velocityX;  
 if (ballY < bottom || ballY > top) velocityY = -velocityY;  
 }  
 }  
 }  
  
 private boolean checkWinCondition() {  
 return Math.*sqrt*((ballX - holeX) \* (ballX - holeX) + (ballY - holeY) \* (ballY - holeY)) < holeRadius;  
 }  
  
 public static void main(String[] args) {  
 new GolfGame().run();  
 }  
}

Приложение Б

**Тесты работы программы:**

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 8 Главное меню игры

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 9 Рекорды

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 10 Информация об игре

Изображение выглядит как снимок экрана, диаграмма, Графическое программное обеспечение, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 11 Первый уровень

Изображение выглядит как снимок экрана, диаграмма, текст, карта

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 12 Все удары использованы

Изображение выглядит как снимок экрана, дисплей, Прямоугольник, прямоугольный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 13 Второй уровень (добавление песка)

Изображение выглядит как снимок экрана, диаграмма, Графическое программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 14 Третий уровень

Изображение выглядит как снимок экрана, Графическое программное обеспечение, Мультимедийное программное обеспечение, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 15 Четвертый уровень

Изображение выглядит как снимок экрана, прямоугольный, диаграмма, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 16 Пятый уровень

Изображение выглядит как снимок экрана, пиксель, диаграмма, План

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 17 Шестой уровень – повышение сложности

Изображение выглядит как снимок экрана, диаграмма, Графическое программное обеспечение, прямоугольный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 18 Седьмой уровень

Изображение выглядит как снимок экрана, Графическое программное обеспечение, диаграмма, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 19 Восьмой уровень – значительное повышение сложности

Изображение выглядит как снимок экрана, Графическое программное обеспечение, Мультимедийное программное обеспечение, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 20 – Повышение сложности с помощью новой механики: прохождение мячика “пиксель в пиксель”

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, Графическое программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 21 Увеличенный по длине уровень с механикой уровня 9

Изображение выглядит как текст, снимок экрана, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 22 Пискель-арт и финал игры

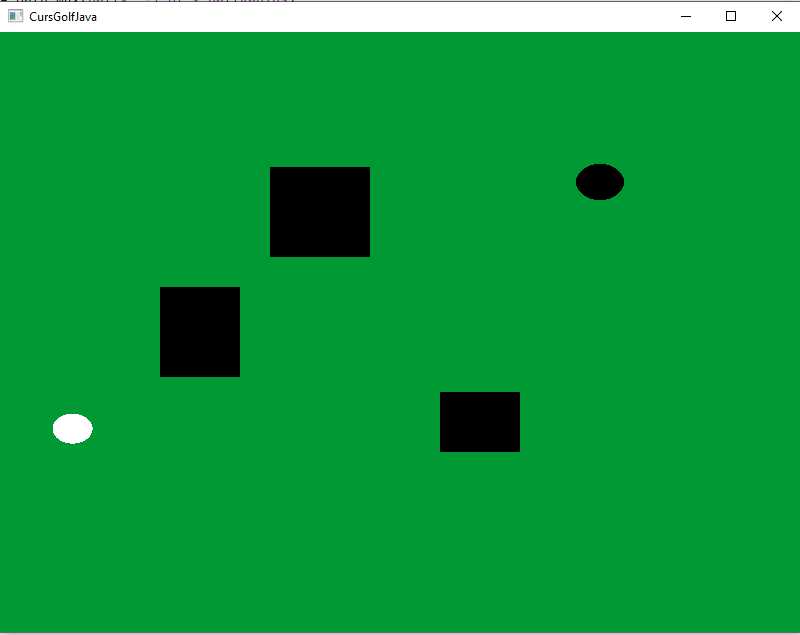


Рисунок 23 Прототип игры на java