МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»   
КАФЕДРА 43

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| канд. техн. наук |  |  |  | А.В.Туманова | |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия | |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ | | | | |
| ИГРА «КРУТОЙ ПИРАТСКИЙ КОРАБЛЬ ПЛАВАЕТ И ПЫТАЕТСЯ ВЫЖИТЬ ИЗ ПОСЛЕДНИХ СИЛ 2D И СОКРОВИЩА» С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ SFML | | | | |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ | | | | |
|  | | | | |
|  | | | | |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4136 |  |  |  | Якшиин С.Е. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**СОДЕРЖАНИЕ**

**1) Постановка задачи.................................................................................................2**

**1.1) Задача..........................................................................................................2**

**1.2) Реализация.................................................................................................2**

**2) Описание структур данных.................................................................................2**

**3) Описание программы и функций.......................................................................3**

**3.1) Глобальные переменные.........................................................................3**

**3.2) Функции.....................................................................................................4**

**3.2.1) create\_core()** и **create\_chest().......................................................4**

**3.2.2) is\_BDUM()** и **is\_COLLECTED().................................................5**

**3.2.3) diff\_up()..........................................................................................5**

**3.2.4) main(). Part I.................................................................................6**

**3.2.5) main() Part II.................................................................................6**

**4) Результаты тестирования....................................................................................6**

**5) Заключение.............................................................................................................9**

**6) Листинг программы............................................................................................10**

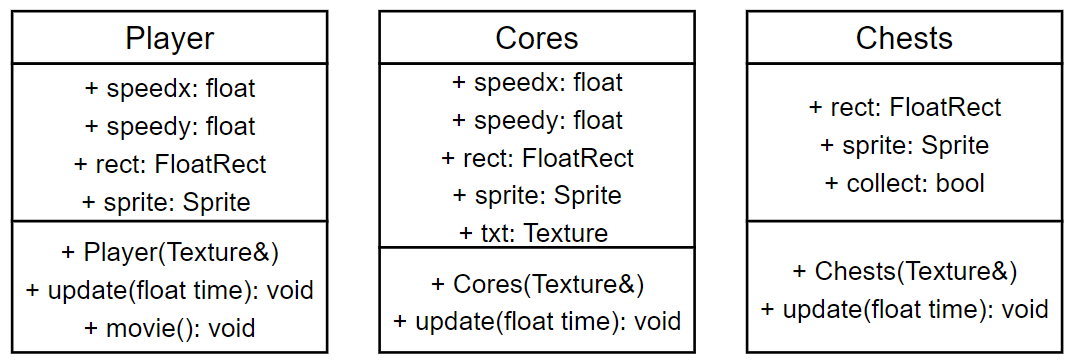
**1. Постановка задачи.**

**1.1)** Задачей курсового проекта является создание игры для настольных пк и ноутбуков под управлением Windows OS. Целью игры является набрать как можно очков, собирая сундуки с сокровищами, внезапно всплыващие в морских глубин, попутно уворачиваясь от вражеских снарядов, которые летят со всех сторон. По мере сбора сундуков сложность игры увеличивается засчет более быстрого появления всё новых снарядов.

**1.2)** Для реализации задумки используется свободная кроссплатформенная мультимедийная библиотека SFML, позволяющая работать с графикой и создавать в том числе игры.

**2. Описание структур данных.**

Для реализации проекта необходимы 3 класса:



Class Player - класс игрока, в данном случае корабль, странствующий меж волнами в поисках сокровищ;

Class Cores - класс препятсвий или помех, в игре представлены как пушечные ядра:

Class Chests - класс игровых очков, сундуков, подбирая которые игрок может отслеживать свой прогресс.

Каждый из классов содержит следующие поля и методы:

**speedx, speedy** - скорость по осям х и у соответсвенно. Нужны для вычисления координат для перемещения по игровому полю. В классе Chests эти поля равны 0 (отсутствуют), т.к. объекты этого класса неподвижны на игровом поле.

**rect** - каркас для спрайта, задает размеры взаимодействия.

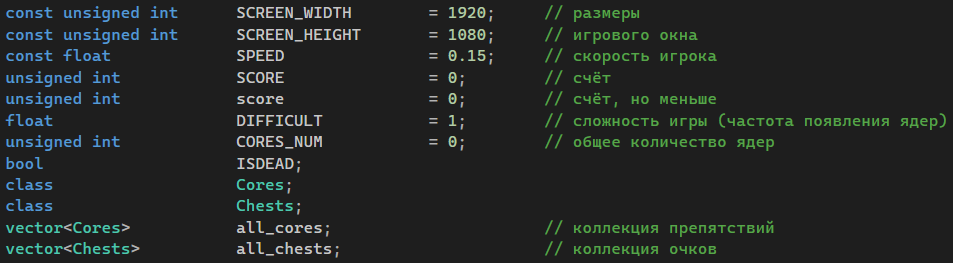
**sprite** - позволяет объектам взаимодействовать с игроком, посредством тектур, и друг с другом, посредством каркаса rect

**update(time)** - основная функция, которая обновляет положение объектов на игровом поле с каждой иттерацией главного цикла с течением времени. На вход функции поступает время в микросекундах и умножается на текущую скорость, таким образом по школьной формуле мы получаем расстояние, а в нашем случае пространственные координаты, заданные в двумерном декартовом пространстве, где по оси Х - ширина экрана в рх, по оси У - высота экрана в рх. Примечание: Точка (0:0) находится в левом верхнем углу, положительное направление для оси Х - вправо, для У - вниз.

**3. Описание программы и функций.**

Программу можно разделить на 4 условные части:

**3.1) Глобальные переменные:**



Для хранения всех объектов классов Cores и Chests используются векторы из стандартной библиотеки STL.

**3.2) Функции:**

В проекте используется как ООП, так и функциональное программирование. Здесь будут описаны функции, которые не являются методами классов.

**3.2.1)** **create\_core()** и **create\_chest()** очень похожи друг на друга. Они принимают на вход указатель на объект соответсвующего класса, создают по этому адресу экземпляр и присваюивают ему какие-нибдь свойства, например место появления и скорость (в случае с create\_core() при создании экземпляра объекта класса ему сразу присваивается начальная скорость и направление. На самом деле ядру присваиваются 2 скорости - скорость по оси х и по оси у. Эти скорости вычислены особым образом так, чтобы при векторном сложении их результирующая направляла ядро к текущему местоположению кораблика на карте. Вот как это выглядит:



Из координаты ядра (которое спавнится по периметру карты) вычитается координата кораблика по х и по у. Таким образом при сложении эти скорости компенсируют друг друга и направляют ядро точно к кораблю. Число 6000 найдено эксперементально и призвано регулировать скорость снарядов, чтобы те не летели слишком быстро), после чего добавляют этот экземпляр в ранее созданный вектор. Для создания препятствий нет никаких ограничейни по их количеству, этот показатель регулируется неявно, периодом между спавном объектов и скоростью их полёта через игровое поле, ведь будучи за его предемали объекты уничтожаются и стираются из памяти, дабы избежать утечки ресурсов. Примечание: Так как начальная скорость ядра зависит от текущих координат кораблика и помножается на коэфициент, она не имеет поправки по разбросу скорости. Этот разброс существует и зависит от того, насколько далеко от кораблика появилось ядро. Чем больше расстояние между двумя объектами, тем больше будет вычисленное произведение скоростей, то бишь вектор, а значит итоговое значение будет больше, чем когда расстояние между объектами маленькое. Эта проблема решается переходои к относительным величинам, например отношением вычисленной скорости к одной из текущих координат игрока. Но в таком случае подплыв близко к краю карты у игрока почти нет шансов увернуться от возникшего рядом с ним снаряда, что, по-моему мнению, вносит дизбаланс в игру. Поэтому результат вычислений было решено оставить в абсолютных величинах.

В отличии от препятствий, сундуки ограничены по количеству - не более 5 штук одновременно. Они появляются за каждый 6 выпущенных ядер, поэтому с увеличением сложности появляются всё чаще, и такое условие не дает заполонить всю карту бонусами. Сундуки уничтожаются при сборе - объект и указатель на него удаляются, освобождая тем самым память и ресурсы машины.

Подытоживая:

**create\_core()** создает неограниченное количество объектов класса Cores, заботится об удалении «невидимых» объектов за пределами игрового поля. Взаимодействует с коллекцией all\_cores, хранящую все текущие препятствия. С другими функциями не взаимодействует.

**create\_core()** создает ограниченное заданное количество объектов класса Chests, заботится об их удалении после «сбора», взаимодействует с коллекцией all\_chests, хранящую все текущие несобранные бонусы. С другими функциями не взаимодествует

**3.2.2) is\_BDUM()** и **is\_COLLECTED()** тоже похожя друг на друга. Эти функции созданны, чтобы обрабатывать взаимодействия с игроком, то есть кораблем. is\_BDUM() формирует хитбокс корабля, в зависимости от направления его движения - прямоугольники размером 24х64 рх или 64х24 рх при движении вертикально и горизонтально, и ступеньки из

3-х квадратов 20х20 при движении по диагонали, и регистрирует попадания ядер в него. С помощью 3-х (4-х) вложенных циклов функция для каждого элемента коллекции проверяет положение каждого пикселя на совпадение с положениям с каждым из пикселем хитбокса корабля (либо с каждым из трёх квадратов, при движении диагонально). Если попадание есть, функция вернет true и игра закончится. is\_COLLECTED() также формирует хитбокс игрока, но более упрощенный - квадрат 64х64 рх, чтобы упростить сбор сундуков. Логика вложенных циклов точно такая же. При взаимодействии объектов сундук исчезнет, а счётчик очков увеличится на 1.

**3.2.3) diff\_up()** служит для постепенного увеличения сложности игры, а если точнее каждые 2 собранных сундука уменьшает интервал появления новых ядер с исходной 1 сек на 0.2 сек вплоть до 0.2 сек. Информация о текущей сложности выводится на экран в виде интервала между спавном новых ядер

**3.2.4) Функция main().** Первая половина. Там объявляются переменные типа Clock, Texture, Sound и тп и загружаются данные с файлов, которые потом используются в функциях и методах.

**3.2.5) Функция main().** Цикл while(). Основной цикл, в котором реализуется игра. В нем на каждой иттерации происходит рассчет координат в зависимости от взаимодействия пользователя с клавиатурой и отрисовка объектов.

**4. Результаты тестирования.**

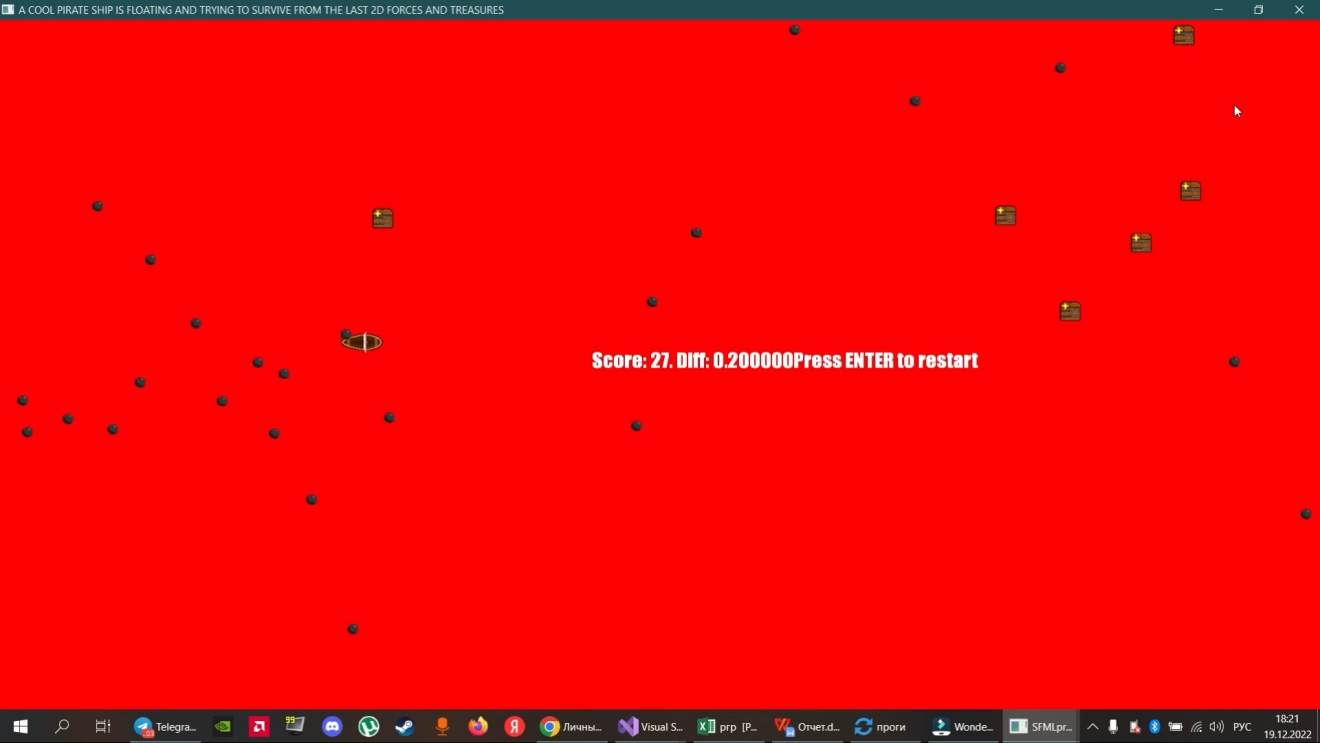
На разных этапах разработки тесты давали различные результаты. С помощью постоянно проводимых тестов получилось исправить многие ошибки спавна препятствий, очков и тд и тп. Окончательные тесты дают понять, что на малопроизводительном железе (либо когда включен режим экономии энергии на ноутбуке) на высокой сложности игра начинает фризить. Решить это можно уменьших таймфрейм, тем самым игра потеряет плавность, но добавит в производительности.

Начало игры:



Сбор\_сундука



Увеличение\_сложностиПоражение

**5. Заключение.**

В ходе курсовой работы была создана игра «КРУТОЙ ПИРАТСКИЙ КОРАБЛЬ ПЛАВАЕТ И ПЫТАЕТСЯ ВЫЖИТЬ ИЗ ПОСЛЕДНИХ СИЛ 2D И СОКРОВИЩА». Игра имеет функцию накопления очков и увеличения сложности. Игра создает новые объекты и очищает память от старых. В игре можно проиграть и начать всё с чистого листа. Присутствует приятное тематическое звуковое сопровождение

**6) Листинг программы:**

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include <iostream>

#include <string>

#include <vector>

#include <cmath>

#include <ctime>

#include <Windows.h>

using namespace sf;

using namespace std;

const unsigned int SCREEN\_WIDTH = 1920; // размеры

const unsigned int SCREEN\_HEIGHT = 1080; // игрового окна

const float SPEED = 0.1; // скорость игрока

unsigned int SCORE = 0; // счёт

unsigned int score = 0; // счёт, но меньше

float DIFFICULT = 1; // сложность игры (частота появления ядер)

unsigned int CORES\_NUM = 0; // общее количество ядер

bool ISDEAD;

class Cores;

class Chests;

vector<Cores> all\_cores; // коллекция препятствий

vector<Chests> all\_chests; // коллекция очков

class Player {

public:

float speedx, speedy; // скорость по осям

FloatRect rect; // рект типа спрайт с текстурой и размеры есть еще чето там вот

Sprite sprite; // спрайт типа фанта

Player(Texture& texture) {

sprite.setTexture(texture); // загрузка текстуры

sprite.setTextureRect(IntRect(0, 0, 64, 64)); // натягивание текстуры на спрайт

rect = FloatRect(SCREEN\_WIDTH / 2, SCREEN\_HEIGHT / 2, 64, 64); // создание ректа

sprite.setPosition(rect.left, rect.top); // расположение спрайта на экране

speedx = speedy = 0; // начальная скорость = 0

ISDEAD = false; // начальное состояние: жив

}

void update(float time) {

rect.left += speedx \* time;

rect.top += speedy \* time;

sprite.setPosition(rect.left, rect.top);

speedy = 0; speedx = 0;

}

void movie() {

if (Keyboard::isKeyPressed(Keyboard::Left) && Keyboard::isKeyPressed(Keyboard::Up) && rect.left > 0 && rect.top > 0 && !ISDEAD)

{ sprite.setTextureRect(IntRect(64 \* 7, 0, 64, 64)); speedx = -(SPEED \* 0.7); speedy = -(SPEED \* 0.7);

}

else if (Keyboard::isKeyPressed(Keyboard::Right) && Keyboard::isKeyPressed(Keyboard::Up) && rect.left < SCREEN\_WIDTH - 64 && rect.top > 0 && !ISDEAD)

{ sprite.setTextureRect(IntRect(64 \* 1, 0, 64, 64)); speedx = (SPEED \* 0.7); speedy = -(SPEED \* 0.7);

}

else if (Keyboard::isKeyPressed(Keyboard::Down) && Keyboard::isKeyPressed(Keyboard::Right) && rect.left < SCREEN\_WIDTH - 64 && rect.top < SCREEN\_HEIGHT - 64 && !ISDEAD)

{ sprite.setTextureRect(IntRect(64 \* 3, 0, 64, 64)); speedx = (SPEED \* 0.7); speedy = (SPEED \* 0.7);

}

else if (Keyboard::isKeyPressed(Keyboard::Down) && Keyboard::isKeyPressed(Keyboard::Left) && rect.top < SCREEN\_HEIGHT - 64 && rect.left > 0 && !ISDEAD)

{ sprite.setTextureRect(IntRect(64 \* 5, 0, 64, 64)); speedx = -(SPEED \* 0.7); speedy = (SPEED \* 0.7);

}

else if (Keyboard::isKeyPressed(Keyboard::Left) && rect.left > 0 && !ISDEAD) {

sprite.setTextureRect(IntRect(64 \* 6, 0, 64, 64)); speedx = -SPEED;

}

else if (Keyboard::isKeyPressed(Keyboard::Right) && rect.left < SCREEN\_WIDTH - 64 && !ISDEAD) {

sprite.setTextureRect(IntRect(64 \* 2, 0, 64, 64)); speedx = SPEED;

}

else if (Keyboard::isKeyPressed(Keyboard::Up) && rect.top > 0 && !ISDEAD) {

sprite.setTextureRect(IntRect(64 \* 0, 0, 64, 64)); speedy = -SPEED;

}

else if (Keyboard::isKeyPressed(Keyboard::Down) && rect.top < SCREEN\_HEIGHT - 64 && !ISDEAD) {

sprite.setTextureRect(IntRect(64 \* 4, 0, 64, 64)); speedy = SPEED;

}

}

};

class Cores {

public:

float speedx, speedy; // скорость по осям

FloatRect rect; // рект типа спрайт с текстурой и размеры есть еще чето там вот

Sprite sprite; // спрайт типа фанта

Texture txt;

Cores(Texture& texture) {

txt = texture;

sprite.setTexture(txt);

sprite.setTextureRect(IntRect(0, 0, 16, 16));

bool left = rand() % 2, top = rand() % 2; // рандомный спавн

if (left) { if (top) { rect = FloatRect(rand() % SCREEN\_WIDTH, 0, 64, 64); } // движущихся

else { rect = FloatRect(SCREEN\_WIDTH, rand() % SCREEN\_HEIGHT, 64, 64); }} // препятствий

else { if (top) { rect = FloatRect(rand() % SCREEN\_WIDTH, SCREEN\_HEIGHT, 64, 64);} // по периметру

else { rect = FloatRect(0, rand() % SCREEN\_HEIGHT, 64, 64); }} // карты

sprite.setPosition(rect.left, rect.top); speedx = speedy = 0; // установка ядра

}

void update(float time, Player& ship) {

if (!ISDEAD) { // самонаведение (недоделанное)

rect.left += -(rect.left - 32 - ship.rect.left) / 6000 \* time;

rect.top += -(rect.top - 32 - ship.rect.top) / 6000 \* time;

}

else { speedy = 0; speedx = 0; }

sprite.setPosition(rect.left, rect.top);

}

void update(float time) {

if (!ISDEAD) { // прямолинейное движение

rect.left += speedx \* time;

rect.top += speedy \* time;

}

else { speedy = 0; speedx = 0; }

sprite.setPosition(rect.left, rect.top);

}

};

class Chests {

public:

FloatRect rect; // рект типа спрайт с текстурой и размеры есть еще чето там вот

Sprite sprite; // спрайт типа фанта

bool collect; // флаг собран или нет

Chests(Texture& texture) {

collect = false;

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(0, 0, 32, 32));

rect = FloatRect(rand() % (SCREEN\_WIDTH - 32), rand() % (SCREEN\_HEIGHT - 32), 32, 32);

sprite.setPosition(rect.left, rect.top);

}

void update() {if (!collect) { sprite.setPosition(rect.left, rect.top); }}

};

void create\_core(Clock& clock, Texture& tCore, Player& ship, Cores\*& pCore) {

if (clock.getElapsedTime().asSeconds() > DIFFICULT && !ISDEAD) {

pCore = new Cores(tCore);

pCore->speedx = -(pCore->rect.left - 32 - ship.rect.left) / 6000; // рассчет скорости и направление

pCore->speedy = -(pCore->rect.top - 32 - ship.rect.top) / 6000; // по х и у к центру кораблика

all\_cores.push\_back(\*pCore); ++CORES\_NUM; int i = 0;

for (auto& el : all\_cores) {

if ((el.rect.left > 1920 || el.rect.left < 0) && (el.rect.top > 1080 || el.rect.top < 0)) { all\_cores.erase(all\_cores.begin() + i); }

else { ++i; }

}

clock.restart();

}

}

void create\_chest(Clock& clock, Texture& tChest, Chests\*& pChest) {

if (CORES\_NUM > 5 && all\_chests.size() <= 5 && !ISDEAD) {

pChest = new Chests(tChest);

all\_chests.push\_back(\*pChest);

clock.restart();

CORES\_NUM = 0;

}

int i = 0;

for (auto& el : all\_chests) {

if (el.collect) { all\_chests.erase(all\_chests.begin() + i); }

else { ++i; }

}

}

void diff\_up() {

if (DIFFICULT > 0.4 && score % 2 == 0 && score != 0) { DIFFICULT -= 0.2; score = 0; }

}

bool is\_BDUM(Player& ship, Texture& tBoom) {

if (Keyboard::isKeyPressed(Keyboard::Left) && Keyboard::isKeyPressed(Keyboard::Up))

{

for (int k = 3; k >= 0; --k) {

for (int i = ship.rect.left + 11 + 8 \* (3 - k); i < (ship.rect.left + ship.rect.width) - 11 - 8 \* k; ++i) {

for (int j = ship.rect.top + 11 + 8 \* (3 - k); j < (ship.rect.top + ship.rect.height) - 11 - 8 \* k; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

}

else if (Keyboard::isKeyPressed(Keyboard::Right) && Keyboard::isKeyPressed(Keyboard::Up))

{

for (int k = 3; k >= 0; --k) {

for (int i = ship.rect.left + 11 + 8 \* k; i < (ship.rect.left + ship.rect.width) - 11 - 8 \* (3 - k); ++i) {

for (int j = ship.rect.top + 11 + 8 \* (3 - k); j < (ship.rect.top + ship.rect.height) - 11 - 8 \* k; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

}

else if (Keyboard::isKeyPressed(Keyboard::Down) && Keyboard::isKeyPressed(Keyboard::Right))

{

for (int k = 3; k >= 0; --k) {

for (int i = ship.rect.left + 11 + 8 \* (3 - k); i < (ship.rect.left + ship.rect.width) - 11 - 8 \* k; ++i) {

for (int j = ship.rect.top + 11 + 8 \* (3 - k); j < (ship.rect.top + ship.rect.height) - 11 - 8 \* k; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

}

else if (Keyboard::isKeyPressed(Keyboard::Down) && Keyboard::isKeyPressed(Keyboard::Left))

{

for (int k = 3; k >= 0; --k) {

for (int i = ship.rect.left + 11 + 8 \* k; i < (ship.rect.left + ship.rect.width) - 11 - 8 \* (3 - k); ++i) {

for (int j = ship.rect.top + 11 + 8 \* (3 - k); j < (ship.rect.top + ship.rect.height) - 11 - 8 \* k; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

}

else if (Keyboard::isKeyPressed(Keyboard::Up))

{

for (int i = ship.rect.left + 18; i < (ship.rect.left + ship.rect.width) - 20; ++i) {

for (int j = ship.rect.top + 1; j < (ship.rect.top + ship.rect.height) - 1; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

else if (Keyboard::isKeyPressed(Keyboard::Down)) {

for (int i = ship.rect.left + 18; i < (ship.rect.left + ship.rect.width) - 20; ++i) {

for (int j = ship.rect.top + 1; j < (ship.rect.top + ship.rect.height) - 1; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

else if (Keyboard::isKeyPressed(Keyboard::Left)) {

for (int i = ship.rect.left + 1; i < (ship.rect.left + ship.rect.width) - 1; ++i) {

for (int j = ship.rect.top + 18; j < (ship.rect.top + ship.rect.height) - 20; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

else if (Keyboard::isKeyPressed(Keyboard::Right)) {

for (int i = ship.rect.left + 1; i < (ship.rect.left + ship.rect.width) - 1; ++i) {

for (int j = ship.rect.top + 18; j < (ship.rect.top + ship.rect.height) - 20; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

return true;

}

}

}

}

}

else {

for (int i = ship.rect.left + 21; i < (ship.rect.left + ship.rect.width) - 21; ++i) {

for (int j = ship.rect.top + 21; j < (ship.rect.top + ship.rect.height) - 21; ++j) {

for (auto& el : all\_cores) {

if (((int)el.rect.left + 8 == i && (int)el.rect.top + 8 == j)) {

el.sprite.setTexture(tBoom);el.sprite.setTextureRect(IntRect(0, 0, 32, 32)); return true;

}

}

}

}

return false;

}

return false;

}

bool is\_COLLECTED(Player& ship) {

for (int i = ship.rect.left + 10; i < (ship.rect.left + ship.rect.width) - 10; ++i) {

for (int j = ship.rect.top + 10; j < (ship.rect.top + ship.rect.height) - 10; ++j) {

for (auto& el : all\_chests) {

if ((int)el.rect.left + 16 == i && (int)el.rect.top + 16 == j) {

el.collect = true; return true;

}

}

}

}

return false;

}

SoundBuffer shootBuffer;

Sound shoot;

int main() {

srand(time(0)); RenderWindow window(sf::VideoMode(SCREEN\_WIDTH, SCREEN\_HEIGHT), "A COOL PIRATE SHIP IS FLOATING AND TRYING TO SURVIVE FROM THE LAST 2D FORCES AND TREASURES");

//-----------------------------------------------------------------------------------------------------------

Clock clock; // часы для отвзяки от мощности пк

Clock cores\_clock; // часы для спавна ядер

Clock chests\_clock; // часы для спавна сундуков

//-----------------------------------------------------------------------------------------------------------

SoundBuffer buffer;

buffer.loadFromFile("music.mp3"); // загрузил крутую музычку

Sound sound;

sound.setBuffer(buffer);

sound.setVolume(30);

sound.play();

shootBuffer.loadFromFile("boom.mp3");

shoot.setBuffer(shootBuffer);

SoundBuffer holeraBuffer;

holeraBuffer.loadFromFile("holera.mp3");

Sound holera(holeraBuffer);

//-----------------------------------------------------------------------------------------------------------

Font font; // загрузка шрифта для текста

if (!font.loadFromFile("impact.ttf")) { cout << "ERROR FONT!!!"; };

Text text; text.setFont(font);

string report;

//-----------------------------------------------------------------------------------------------------------

Texture tship; tship.loadFromFile("ship.png"); // текстура кораблика

Player ship(tship); // кораблик (игрок)

Texture tWater; tWater.loadFromFile("water.jpg"); // текстура воды (фона)

Sprite sWater; sWater.setTexture(tWater); // спрайт воды

sWater.setPosition(0, 0);

Texture tCore; tCore.loadFromFile("core.png"); // текстура ядра (препятствия)

Cores\* pCore;

Texture tBoom; tBoom.loadFromFile("boom.png");

Texture tChest; tChest.loadFromFile("chest.png"); // текстура сундука (очка счёта)

Chests\* pChest;

//-----------------------------------------------------------------------------------------------------------

while (window.isOpen()) {

{Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

}}

float time = clock.getElapsedTime().asMicroseconds();

clock.restart(); time = time / 250; // скорость обновления анимации

//-----------------------------------------------------------------------------------------------------------

// действия:

create\_core(cores\_clock, tCore, ship, \*&pCore);

diff\_up();

create\_chest(chests\_clock, tChest, \*&pChest);

ship.movie();

if (is\_BDUM(ship, tBoom)) { shoot.play(); holera.play(); ISDEAD = true; }

if (is\_COLLECTED(ship)) { ++SCORE; ++score; }

string score = "Score: ", diff = ". Diff: ";

report += score += to\_string(SCORE) += diff += to\_string(DIFFICULT);

text.setString(report);

//-----------------------------------------------------------------------------------------------------------

// update-функции:

ship.update(time); // обновление кораблика

for (auto& el : all\_cores) { el.update(time); } // обновление всех ядер

for (auto& el : all\_chests) { el.update(); } // обновление всех сундуков

//-----------------------------------------------------------------------------------------------------------

// отрисовка:

if (!ISDEAD) { window.draw(sWater); } // отрисовка фона во время игры

else { window.clear(Color::Red); } // и при поражении

window.draw(ship.sprite); // отрисовка кораблика

for (auto& el : all\_cores) { window.draw(el.sprite); } // отрисовка всех ядер

for (auto& el : all\_chests) { window.draw(el.sprite); } // отрисовка всех сундуков

window.draw(text); // отрисовка очков

report.clear(); // очистка строки

window.display();

}

}