Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционный системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

Программное средство «Mouse recorder»

Студент: гр. 451005

Тонко И. В.

Руководитель:

Марина И. М.

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2015 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту    Тонко Игорю Витальевичу––––––––––––––––––––––––

1. Тема работы   Программное средство «Mouse recorder»

2. Срок сдачи студентом законченной работы––15.12.2015

3. Исходные данные к работе ———

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение;

1. Анализ предметной области;

2. Разработка программного средства;

3. Руководство пользователя;

Заключение.

4. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема алгоритма работы системы

5. Консультант по курсовой работе Марина И. М.

6. Дата выдачи задания 09.09.2015 г.–––––––––––––––––––––––   –

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1,2 к 20.09.2015 – 15 % готовности работы;

разделы 3, 4 к 15.10.2015 – 30 % готовности работы;

разделы 5, 6, к 07.11.2015 – 60 % готовности работы;

раздел 7, 8, 9 к 17.11.2015 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 24.11.2015- 100 % готовности работы.

Защита курсового проекта с 15 декабря 2015.

РУКОВОДИТЕЛЬ - Марина И. М.

(подпись)

Задание принял к исполнению –––\_\_\_\_–– И. В. Тонко    09.09.2015 г.

(дата и подпись студента)

**СОДЕРЖАНИЕ**

Введение 4

1 Анализ предметной области 5

1.1 Анализ аналогов программного средства 5

2 Разработка программного средства 7

2.1 Общая информация 7

2.2 Графические инструменты 7

2.3 Визуальное представление 8

2.3.1 Платформа 8

2.3.2 Игровое поле 8

2.4 Игровая механика 10

2.4.1 Вспомогательные массивы 10

2.4.2 Вспомогательные глобальные переменные 11

2.4.3 Дополнительные эффекты 11

2.4.4 Физика шарика 14

2.4.5 Взаимодействие шарика с платформой 15

2.4.6 Взаимодействие шарика с блоками 16

2.4.7 Генерация дополнительных эффектов 16

2.4.8 Отображение текстовой информации 16

2.5 Описание блок-схемы 19

3 Руководство пользователя 21

3.1 Инструкция по использованию 21

Заключение 24

Список использованных источников 25

Приложение А. Исходный код программы 26

Приложение Б. Блок-схема главного цикла программы 50

**ВВЕДЕНИЕ**

В современном мире наиболее развивающаяся отрасль – это отрасль компьютеров. Еще несколько десятилетий назад все персональные компьютеры можно было пересчитать по пальцам. Однако, сейчас их количество возросло настолько, что в каждой семье есть один, а может быть и более компьютеров. В наше время информационные технологии занимают центральное место в интеллектуализации общества, развитии его системы образования и культуры.

Windows API ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) application programming interfaces) — общее наименование целого набора базовых функций [интерфейсов программирования приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) операционных систем семейств [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows) корпорации «[Майкрософт](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B9%D0%BA%D1%80%D0%BE%D1%81%D0%BE%D1%84%D1%82)». Является самым прямым способом взаимодействия приложений с Windows.

Windows API спроектирован для использования в языке [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) для написания [прикладных программ](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), предназначенных для работы под управлением операционной системы MS Windows. Работа через Windows API — это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ.  Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си. Все языки программирования, способные вызывать такие функции и оперировать такими типами данных в программах, исполняемых в среде Windows, могут пользоваться этим API. В частности, это языки [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Pascal](https://ru.wikipedia.org/wiki/Pascal), [Visual Basic](https://ru.wikipedia.org/wiki/Visual_Basic) и многие другие.

1. Анализ предметной области
   1. Анализ аналогов программного средства

**«**Mouse Recorder Pro**»** — приложение, предназначенное для автоматизации рутинных задач. Эта программа позволяет записывать действия, выполняемые пользователем на компьютере (движения мышки, нажатия кнопок клавиатуры и т.д.) и затем воспроизводить их.

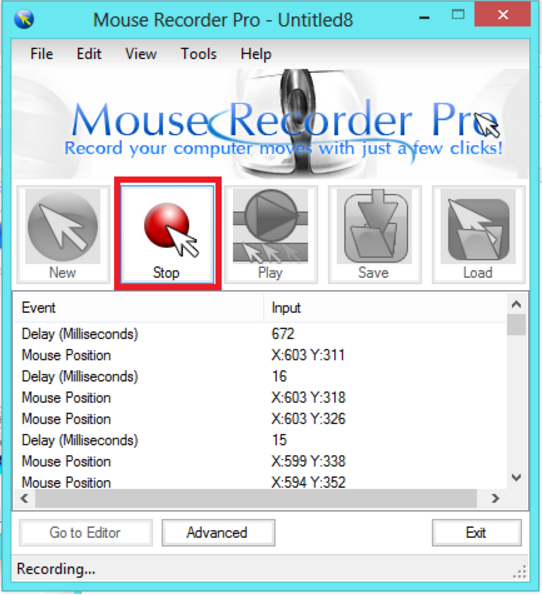


Рисунок 1.1 - Mouse Recorder Pro

Пользователь нажимает кнопку записи и производит мышкой какие-либо действия, а затем нажимает кнопку F12 для остановки записи. После этого пользователь может ознакомиться со списком его действий, которые записала программа и при необходимости удалить лишнее.

Теперь пользователь может запускать записанный макрос, чтобы избавить себя от необходимости выполнять действия вручную (макрос может воспроизводиться и автоматически - в заданное пользователем время). Программа умеет запоминать не только движения мышки и нажатия её кнопок, но и нажатия на кнопки клавиатуры. Mouse Recorder Pro отличается дружественным интерфейсом, имеет небольшой размер и практически не потребляет системные ресурсы.

**«**Auto Macro Recorder**»** — еще одно приложение, предназначенное для записи и воспроизведения действий пользователя.

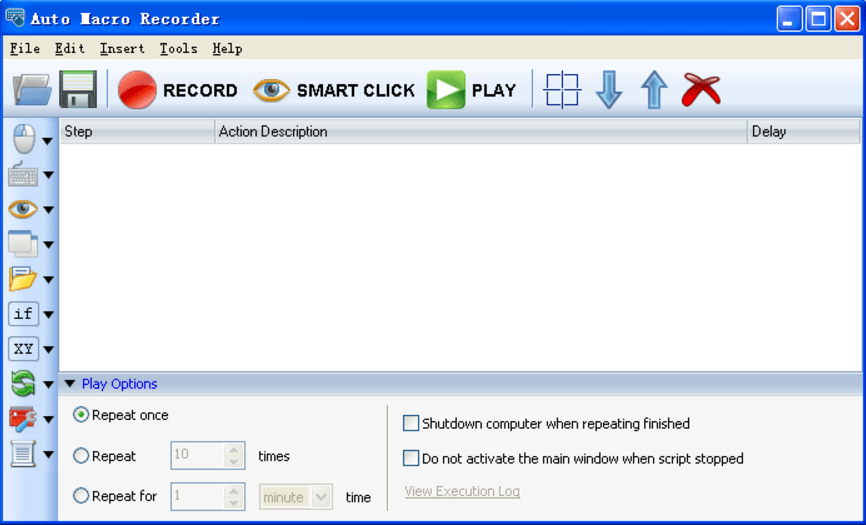


Рисунок 1.2 - Auto Macro Recorder

* 1. Постановка задачи

Задачей данной курсовой работы является создание упрощенной версии приложения для записи и проигрывания пользовательских нажатий левой кнопки мыши.

Реализовать следующие функции: запись макроса в файл, чтение макроса из файла, проигрывание макроса через определенные пользователем промежутки времени.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
   1. Общая информация

При проектировании программы была выбрана методика нисходящего проектирования: сначала была проработана основная концепция программы (получение, обработка и воспроизведения кликов мыши).

В рамках данного курсового проекта планируется разработать программное средство.

Для разработки программного средства использовать язык программирования C++. Среда для разработки – Microsoft Visual Studio 2015.

* 1. Визуальное представление

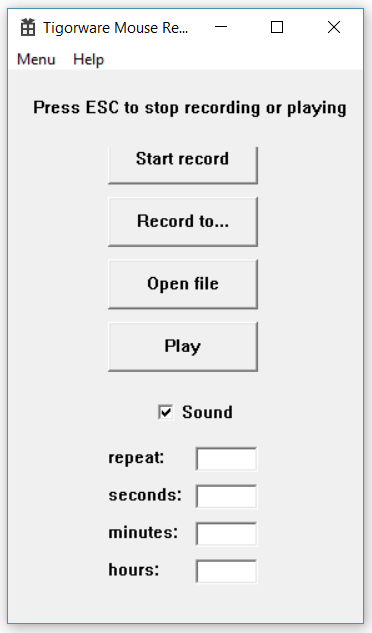


Рисунок 2.1 - Внешний вид приложения

* 1. Игровая механика
     1. Вспомогательные переменные

Программа использует глобальные переменные:

* Все дескрипторы окон (HWND).
* HINSTANCE hInst - главная сущность.
* UINT\_PTR tmr, playTmr - указатели на таймеры.
* POINT CrPos - точка нажатия мыши.
* int posX, posY - координаты точки нажатия мыши.
* HANDLE hFile - дескриптор файла.
* SYSTEMTIME startTime, lastClickTime, clickTime - переменные, хранящие время наступления определенных событий.
* LONG difTime, delayTime - разница временных промежутков в миллисекундах.
* string fileName - имя файла, по умолчанию "temp.m".
  1. Описание блок-схемы

На блок-схеме представлен алгоритм главного цикла программы. При работе с графическим движком SFML все графические изменения происходят пока открыто окно. Алгоритм цикла "Пока открыто окно":

1. Получить время. С помощью средств движка SFML на каждой итерации вычисляется время, прошедшее с прошлого измерения времени в микросекундах.
2. Обработка нажатия клавиш. Набор действий, реагирующих на нажатия клавиш клавиатуры и выполняющих соответствующие преобразования.
3. Обработка движения курсора. Поскольку платформа управляется с помощью мыши, необходимо обрабатывать перемещения курсора, перемещая соответственно платформу.
4. Обработка нажатия кнопки мыши. Левая кнопка мыши отвечает за начало игры, а также за стрельбу из лазерной пушки.
5. Поимка бонусов. Проверяет, словил ли игрок выпавший блок с дополнительным эффектом.
6. Цикл, обрабатывающий движение каждого шарика, имеющегося на игровом поле. Если шарик сталкивается с блоком или со стенкой, выполняются соответствующие преобразования. Также если таймер движения становится больше, чем заранее определенная игровая скорость (частота обновления игрового поля в микросекундах), выполняется изменение координат шариков.
7. Если на игровом поле имеется блок с дополнительным эффектом, изменяются его координаты.
8. Если таймер движения больше, чем игровая скорость, таймер движения обнуляется.
9. Очистка окна.
10. Прорисовка всех компонентов: карты блоков, шариков, блоков-бонусов, платформы и текстовой информации.
11. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данная программа предназначена для использования в целях автоматизации рутинных задач. Приложение поможет выполнить повторяющиеся нажатия левой кнопки мыши без участия пользователя. Приложение удобно и просто в применении, поэтому каждый пользователь сможет с лёгкостью записать и воспроизвести нужный макрос.

* 1. Инструкция по использованию

Кликните двойным щелчком на значок с надписью «Mouse Recorder». Программа запустится на вашем компьютере.

Запустив приложение, пользователь увидит главное окно.

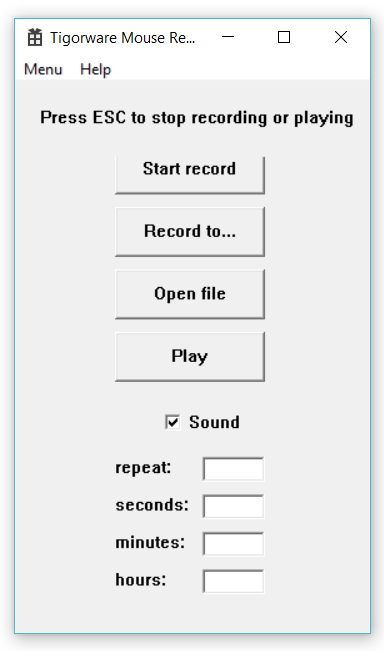


Рисунок 3.1 - Начало работы

Запись нажатий левой кнопки мыши начнется после нажатия кнопки "Start record". Это нажатие не будет записано. Запись будет происходить до нажатия клавиши ESC. Если в поле "Sound" стоит галочка, каждое нажатие будет обозначено коротким звуковым сигналом.

По умолчанию, макрос записывается в файл temp.m. При желании, пользователь может изменить файл для записи, нажав кнопку "Record to...", чтобы выбрать или создать файл, куда будет записан макрос.

Чтобы выбрать файл с записанным макросом, пользователю следует нажать кнопку "Open file". После этого откроется стандартное окно выбора файлов.

.

Рисунок 3.2 - Выпавший бонус

При желании, игрок может словить его платформой, тогда бонус активируется и выполнит преобразования. Если игрок не будет ловить бонус, то он просто исчезнет.

Игра сопровождена звуковыми эффектами. Характерные звуки отскоков мяча, разрушения блоков, выпадения и поимки бонусов делают игру более насыщенной.

Когда пользователь теряет все шарики на карте, у него отнимается жизнь, нейтрализуются все бонусы и шарик закрепляется на платформе, как в начале игры. Если количество жизней станет равно нулю, игра закончится.



Рисунок 3.3 - Конец игры

Чтобы сыграть еще раз, необходимо перезапустить приложение.

**ЗАКЛЮЧЕНИЕ**

Ещё с древних времён люди находили различные способы развлечения. Один из них – игры. Сложно представить человека, который ни разу не играл в игры, будь то компьютерные или настольные. Цель данной курсовой работы – разработать игровое программное средство, которое поможет пользователю провести время с удовольствием.

Результатом данной курсовой работы является игровое приложение, созданное в интегрированной среде разработки Microsoft Visual Studio 2010, с помощью графического движка SFML 2.3.2.

В ходе разработки было изучено большое количество литературы из различных источников. Были совершенствованы знания о работе с графическими и звуковыми компонентами.

Проделанная работа позволяет в полной мере раскрыть у автора наличие базовых знаний и основных навыков в области программирования. А также знание игровой механики.

Возможно дальнейшее усовершенствование проекта:

* улучшить графику;
* добавить главное меню;
* добавить новые типы блоков и бонусов;
* внести в игру новые для данного жанра возможности.

Игровое приложение «Арканоид» можно продолжать развивать, получая новые, все более интересные игры, которые никогда не перестанут притягивать к себе внимание любителей жанра.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Википедия - свободная энциклопедия [Электронный ресурс]. - История компьютерных игр, Arcanoid. Режим доступа : http://www.pravo.by/ . Дата доступа : 15.10.2015
2. Интернет-портал Андрея Букреева [Электронный ресурс] - SFML, разработка 2D игр, язык С++. Режим доступа: http://kychka-pc.ru/ . Дата доступа: 20.11.2015
3. Макс Шлее, Профессиональное программирование на С++ – М.: БХВ-Петербург, 2010. – 883 с.
4. Федоренко Ю.П., Алгоритмы и программы на C++ Builder –М.: ДМК Пресс,2010. –544 с.
5. Мартин.Р., Чистый код. Создание, анализ и рефакторинг: Петербург,2015. –464 с.

Приложение А. Исходный код программы

Main.cpp

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include <sstream>

#include <iostream>

#include <math.h>

#include <malloc.h>

#include <stdlib.h>

#include <windows.h>

#include "map.h"

#include "audio.h"

using namespace sf;

float GetAngle(int offset, int plt\_length)

{

float res;

offset-=plt\_length/2;

res=(float)offset/(plt\_length/2);

return res;

}

struct TBallPosition

{

float x, y;

};

struct TGun

{

bool is\_gun;

int shoot\_timer;

int shoot\_count;

int shoot\_speed;

};

struct TBreaking

{

int count;

int timer;

};

struct TBreakingP

{

float x, y;

int step;

int step\_time;

};

struct TShootPosition

{

float x, y;

};

struct TWindowSize

{

int left, right, top, bot;

};

void AddBall(struct TBallPosition \*b\_pos, int new\_count, float x, float y)

{

b\_pos=(struct TBallPosition \*)realloc(b\_pos, new\_count\*sizeof(struct TBallPosition));

b\_pos[new\_count-1].x=x;

b\_pos[new\_count-1].y=y;

}

struct TBonusPosition

{

float x, y;

int x1, y1, x2, y2;

};

bool BonusMaking(struct TBonusPosition \*bon\_pos, int \*x\_speed, int speed, float \*angle, int block\_x, int block\_y, int \*bonus\_type)

{

int bonus\_chance, add\_life\_change;

srand(unsigned(time(NULL)));

bonus\_chance=rand()%100;

if (bonus\_chance<50)

{

\*bonus\_type=rand()%10+1;

add\_life\_change=rand()%30+1;

if (add\_life\_change==7)

\*bonus\_type=11;

//std::cout<<\*bonus\_type<<"\n";

(\*bon\_pos).x1=(\*bonus\_type-1)\*40+1;

(\*bon\_pos).y1=0;

(\*bon\_pos).x2=39;

(\*bon\_pos).y2=40;

(\*bon\_pos).x=block\_x\*45;

(\*bon\_pos).y=block\_y\*30;

\*x\_speed=speed;

return true;

}

else

return false;

}

int \*x\_speed, \*y\_speed;

struct TBallPosition \*ball\_pos;

float \*angle;

void SplitBall(int \*ball\_count)

{

int nball\_count=\*ball\_count;

for (int x\_ball=0; x\_ball<nball\_count; x\_ball++)

{

(\*ball\_count)++;

ball\_pos=(struct TBallPosition \*)realloc(ball\_pos, (\*ball\_count)\*sizeof(struct TBallPosition));

ball\_pos[\*ball\_count-1].x=ball\_pos[x\_ball].x;

ball\_pos[\*ball\_count-1].y=ball\_pos[x\_ball].y;

x\_speed=(int \*)realloc(x\_speed, \*ball\_count\*sizeof(int));

x\_speed[\*ball\_count-1]=-x\_speed[x\_ball];

y\_speed=(int \*)realloc(y\_speed, \*ball\_count\*sizeof(int));

y\_speed[\*ball\_count-1]=y\_speed[x\_ball];

angle=(float \*)realloc(angle, \*ball\_count\*sizeof(float));

angle[\*ball\_count-1]=angle[x\_ball];

// std::cout<<"y\_speed="<<y\_speed[ball\_count-1]<<"\n";

}

}

int ChangePlatform(int plt\_size, bool increase)

{

int max=400;

int min=50;

if (increase)

{

if(plt\_size<max)

return plt\_size\*2;

else

return plt\_size;

}

else

{

if(plt\_size>min)

return plt\_size/2;

else

return plt\_size;

}

}

int GetPlatformY(int plt\_length)

{

int x=plt\_length/100;

switch(x)

{

case 0:

return 0;

case 1:

return 20;

case 2:

return 40;

case 4:

return 60;

}

}

char NextBlock(char block, bool unstopable)

{

switch(block)

{

case 'g': return ' ';

case 'y': return 'g';

case 'i': return 'g';

case 'r':

if(unstopable)

return ' ';

else

return 'r';

}

}

bool IsEnd()

{

for (int i = 0; i < HEIGHT\_MAP; i++)

for (int j = 0; j < WIDTH\_MAP; j++)

{

if ((TileMap[i][j]=='g')||(TileMap[i][j]=='y')||(TileMap[i][j]=='i'))

return false;

}

return true;

}

int main()

{

FreeConsole();

bool is\_game=false, end\_game=false, is\_pause=false, cursor\_mess=false;

int bx\_speed, by\_speed=2;

int score=0, score\_k=1, lifes=3;

int plt\_pos, plt\_lvl;

int game\_speed=10000, elapsed\_time=0;

int move\_timer=0;

float x\_disp, y\_disp;

int block\_x, block\_y;

int ball\_count=1;

int cur\_ball;

bool is\_bonus=false, unstopable=false;

int bonus\_chance, bonus\_type;

struct TBonusPosition bonus\_pos;

float bangle=1;

int plt\_length;

struct TBreaking breaking\_block;

struct TBreakingP \*bblocks=NULL;

ball\_pos=(struct TBallPosition \*)malloc(ball\_count\*sizeof(struct TBallPosition));

x\_speed=(int\*)malloc(ball\_count\*sizeof(int));

x\_speed[0]=5;

y\_speed=(int\*)malloc(ball\_count\*sizeof(int));

y\_speed[0]=5;

angle=(float\*)malloc(ball\_count\*sizeof(float));

//srand(unsigned(time(NULL)));

struct TWindowSize window\_size;

SoundBuffer sb\_plt, sb\_wall, sb\_ball, sb\_red, sb\_gun, sb\_bblock, sb\_fanfare, sb\_bonus, sb\_invis, sb\_glass, sb\_good, sb\_bad;

Sound sound\_plt, sound\_wall, sound\_ball, sound\_red, sound\_gun, sound\_bblock, sound\_fanfare, sound\_bonus, sound\_invis, sound\_glass, sound\_good, sound\_bad;

sb\_plt.loadFromFile("Sounds/BOING.wav");

sound\_plt.setBuffer(sb\_plt);

sb\_wall.loadFromFile("Sounds/BASSDRUM.wav");

sound\_wall.setBuffer(sb\_wall);

sb\_ball.loadFromFile("Sounds/BYEBALL.wav");

sound\_ball.setBuffer(sb\_ball);

sb\_red.loadFromFile("Sounds/EFFECT.wav");

sound\_red.setBuffer(sb\_red);

sb\_gun.loadFromFile("Sounds/GUNFIRE.wav");

sound\_gun.setBuffer(sb\_gun);

sb\_bblock.loadFromFile("Sounds/BANG.wav");

sound\_bblock.setBuffer(sb\_bblock);

sound\_bblock.setVolume(20.f);

sb\_fanfare.loadFromFile("Sounds/FANFARE.wav");

sound\_fanfare.setBuffer(sb\_fanfare);

sb\_bonus.loadFromFile("Sounds/PEOW!.wav");

sound\_bonus.setBuffer(sb\_bonus);

sb\_invis.loadFromFile("Sounds/ORCHBLAS.wav");

sound\_invis.setBuffer(sb\_invis);

sb\_glass.loadFromFile("Sounds/AO-LASER.wav");

sound\_glass.setBuffer(sb\_glass);

sb\_good.loadFromFile("Sounds/SAUCER.wav");

sound\_good.setBuffer(sb\_good);

sb\_bad.loadFromFile("Sounds/PADEXPLO.wav");

sound\_bad.setBuffer(sb\_bad);

window\_size.left=0;

window\_size.right=1170;

window\_size.top=0;

window\_size.bot=900;

Image map\_image;

map\_image.loadFromFile("Src/green1.png");

Texture map;

map.loadFromImage(map\_image);

Sprite s\_map;

s\_map.setTexture(map);

GenerateMap();

Image bonus\_img;

bonus\_img.loadFromFile("Src/bonus.png");

Texture bonus;

bonus.loadFromImage(bonus\_img);

Sprite s\_bonus;

s\_bonus.setTexture(bonus);

Image gun\_img;

gun\_img.loadFromFile("Src/gun.png");

Texture tgun;

tgun.loadFromImage(gun\_img);

Sprite s\_gun;

s\_gun.setTexture(tgun);

s\_gun.setTextureRect(IntRect(0, 0, 10, 20));

struct TGun gun;

gun.is\_gun=false;

gun.shoot\_count=0;

gun.shoot\_speed=-10;

gun.shoot\_timer=300000;

struct TShootPosition \*shoots=NULL;

int shoot\_timer=0;

Image breaking\_image;

breaking\_image.loadFromFile("Src/green1.png");

Texture breaking;

breaking.loadFromImage(breaking\_image);

Sprite s\_breaking;

s\_breaking.setTexture(breaking);

breaking\_block.count=0;

breaking\_block.timer=100000;

bonus\_pos.x=0;

bonus\_pos.y=0;

bonus\_pos.x1=0;

bonus\_pos.x2=0;

bonus\_pos.y1=0;

bonus\_pos.y2=0;

s\_bonus.setTextureRect(IntRect(0, 0, 0, 0));

s\_bonus.setPosition(bonus\_pos.x, bonus\_pos.y);

RenderWindow window(sf::VideoMode(window\_size.right, window\_size.bot), "T.Igorball v1", sf::Style::Close);

window.setMouseCursorVisible(false);

//ShowWindow(GetConsoleWindow(),SW\_HIDE);

plt\_pos=window\_size.right/2-50;

plt\_lvl=window\_size.bot-50;

plt\_length=100;

//std::cout<<GetPlatformY(plt\_length);

Image plt\_image;

plt\_image.loadFromFile("Src/test\_platform.png");

Texture plt;

plt.loadFromImage(plt\_image);

Sprite s\_plt;

s\_plt.setTexture(plt);

s\_plt.setTextureRect(IntRect(0, GetPlatformY(plt\_length), plt\_length, 19));

s\_plt.setPosition(plt\_pos, plt\_lvl);

ball\_pos[0].x=plt\_pos+43;

ball\_pos[0].y=plt\_lvl-15;

Image ball\_image;

ball\_image.loadFromFile("Src/test\_ball.png");

Texture ball;

ball.loadFromImage(ball\_image);

Sprite s\_ball;

s\_ball.setTexture(ball);

s\_ball.setTextureRect(IntRect(0, 0, 15, 15));

s\_ball.setPosition(ball\_pos[0].x, ball\_pos[0].y);

Clock clock;

Font font;

font.loadFromFile("New\_Zelek Normal.ttf");

Text text("", font, 20);

text.setColor(Color::Red);

//text.setStyle(sf::Text::Bold | sf::Text::Underlined);//жирный и подчеркнутый текст. по умолчанию он "худой":)) и не подчеркнутый

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

}

float time = clock.getElapsedTime().asMicroseconds(); //дать прошедшее время в микросекундах

clock.restart(); //перезагружает время

time = time; //скорость игры

elapsed\_time+=time;

if(elapsed\_time>200000)

{

if (Keyboard::isKeyPressed(Keyboard::Space))

{

//is\_pause=-is\_pause;

if (is\_pause)

is\_pause=false;

else

is\_pause=true;

elapsed\_time=0;

//GenerateMap();

}

}

if (Keyboard::isKeyPressed(Keyboard::Escape))

{

window.close();

}

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

cursor\_mess=false;

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

cursor\_mess=true;

if (!is\_pause)

{

if (event.type == Event::MouseMoved)

{

if (event.mouseMove.x<window\_size.left+(plt\_length/2))

{

s\_plt.setPosition(window\_size.left, plt\_lvl);

plt\_pos=window\_size.left;

cursor\_mess=false;

if (!is\_game)

{

s\_ball.setPosition(window\_size.left+(plt\_length/2)-7, plt\_lvl-15);

ball\_pos[0].x=window\_size.left+(plt\_length/2)-7;

ball\_pos[0].y=plt\_lvl-15;

}

}

else

if (event.mouseMove.x>window\_size.right-(plt\_length/2))

{

s\_plt.setPosition(window\_size.right-plt\_length, plt\_lvl);

plt\_pos=window\_size.right-plt\_length;

cursor\_mess=false;

if (!is\_game)

{

s\_ball.setPosition(window\_size.right-(plt\_length/2)-7, plt\_lvl-15);

ball\_pos[0].x=window\_size.right-(plt\_length/2)-7;

ball\_pos[0].y=plt\_lvl-15;

}

}

else

{

s\_plt.setPosition(event.mouseMove.x-(plt\_length/2), plt\_lvl);

plt\_pos=event.mouseMove.x-(plt\_length/2);

cursor\_mess=false;

if (!is\_game)

{

s\_ball.setPosition(event.mouseMove.x-7, plt\_lvl-15);

ball\_pos[0].x=event.mouseMove.x-7;

ball\_pos[0].y=plt\_lvl-15;

}

}

}

if (event.type == Event::MouseButtonPressed)

if (event.mouseButton.button == sf::Mouse::Button::Left)

{

if ((!is\_game)&&(!end\_game))

{

is\_game=true;

ball\_count=1;

}

else

if (gun.is\_gun)

if (shoot\_timer>gun.shoot\_timer)

{

sound\_gun.play();

shoot\_timer=0;

gun.shoot\_count+=2;

shoots=(struct TShootPosition \*)realloc(shoots, gun.shoot\_count\*sizeof(struct TShootPosition));

shoots[gun.shoot\_count-1].x=plt\_pos+12;

shoots[gun.shoot\_count-1].y=plt\_lvl-20;

shoots[gun.shoot\_count-2].x=plt\_pos+plt\_length-18;

shoots[gun.shoot\_count-2].y=plt\_lvl-20;

}

}

// std::cout<< time<<"\n";

move\_timer+=time;

if (gun.is\_gun)

shoot\_timer+=time;

for (int i=0; i<breaking\_block.count; i++)

{

bblocks[i].step\_time+=time;

if (bblocks[i].step\_time>breaking\_block.timer)

{

bblocks[i].step\_time-=breaking\_block.timer;

bblocks[i].step++;

}

}

//BONUS CATCHING

if (is\_bonus)

{

if ((bonus\_pos.y>=plt\_lvl-40))

{

if ((bonus\_pos.x>=plt\_pos-40) && (bonus\_pos.x<=plt\_pos+plt\_length))

{

score+=100\*score\_k;

switch(bonus\_type)

{

case 1:

{

SplitBall(&ball\_count);

sound\_good.play();

break;

}

case 2:

{

sound\_good.play();

plt\_length=ChangePlatform(plt\_length, true);

s\_plt.setTextureRect(IntRect(0, GetPlatformY(plt\_length), plt\_length, 19));

break;

}

case 3:

{

sound\_bad.play();

plt\_length=ChangePlatform(plt\_length, false);

s\_plt.setTextureRect(IntRect(0, GetPlatformY(plt\_length), plt\_length, 19));

break;

}

case 4:

{

sound\_good.play();

unstopable=true;

break;

}

case 5:

{

sound\_bad.play();

for (int i\_ball=0; i\_ball<ball\_count; i\_ball++)

{

x\_speed[i\_ball]=(abs(x\_speed[i\_ball])/x\_speed[i\_ball])\*7;

y\_speed[i\_ball]=(abs(y\_speed[i\_ball])/y\_speed[i\_ball])\*7;

}

break;

}

case 6:

{

sound\_good.play();

gun.is\_gun=true;

break;

}

case 7:

{

sound\_good.play();

for (int i = 0; i < HEIGHT\_MAP; i++)

for (int j = 0; j < WIDTH\_MAP; j++)

{

if (TileMap[i][j]=='i')

block\_count--;

if (TileMap[i][j]=='r')

block\_count++;

if ((TileMap[i][j]=='i')||(TileMap[i][j]=='r'))

TileMap[i][j]='g';

}

break;

}

case 8:

{

sound\_good.play();

for (int i\_ball=0; i\_ball<ball\_count; i\_ball++)

{

x\_speed[i\_ball]=(abs(x\_speed[i\_ball])/x\_speed[i\_ball])\*3;

y\_speed[i\_ball]=(abs(y\_speed[i\_ball])/y\_speed[i\_ball])\*3;

}

break;

}

case 9:

{

if (lifes<3)

lifes++;

else

score+=1000\*score\_k;

sound\_fanfare.play();

break;

}

case 10:

{

sound\_good.play();

score\_k\*=2;

break;

}

case 11:

{

ball\_count=1;

ball\_pos[0].x=-20;

ball\_pos[0].y=window\_size.bot;

break;

}

}

bonus\_pos.x=0;

bonus\_pos.y=0;

bonus\_pos.x1=0;

bonus\_pos.x2=0;

bonus\_pos.y1=0;

bonus\_pos.y2=0;

bx\_speed=0;

by\_speed=2;

bangle=1;

}

is\_bonus=false;

}

}

if (is\_game)

{

for(cur\_ball=0; cur\_ball<ball\_count; cur\_ball++)

{

if ((ball\_pos[cur\_ball].y>=plt\_lvl-15))

{

if ((ball\_pos[cur\_ball].x>=plt\_pos-15) && (ball\_pos[cur\_ball].x<=plt\_pos+plt\_length+15))

{

sound\_plt.play();

angle[cur\_ball]=GetAngle((int)(ball\_pos[cur\_ball].x-plt\_pos), plt\_length);

x\_speed[cur\_ball]=abs(x\_speed[cur\_ball]);

y\_speed[cur\_ball]=-y\_speed[cur\_ball];

// ball.sprite.setPosition((int)ball\_pos[cur\_ball].x, plt\_lvl-16);

ball\_pos[cur\_ball].y=plt\_lvl-16;

}

else

{

sound\_ball.play();

if (cur\_ball!=ball\_count-1)

{

//std::cout<<"shift\n";

for (int i=cur\_ball; i<ball\_count-1; i++)

{

ball\_pos[i]=ball\_pos[i+1];

x\_speed[i]=x\_speed[i+1];

y\_speed[i]=y\_speed[i+1];

angle[i]=angle[i+1];

}

}

ball\_count--;

if (ball\_count!=0)

{

ball\_pos=(struct TBallPosition \*)realloc(ball\_pos, ball\_count\*sizeof(struct TBallPosition));

x\_speed=(int \*)realloc(x\_speed, ball\_count\*sizeof(int));

y\_speed=(int \*)realloc(y\_speed, ball\_count\*sizeof(int));

angle=(float \*)realloc(angle, ball\_count\*sizeof(float));

}

//END GAME!!!

if (ball\_count==0)

{

// std::cout<<"new ball\n";

score\_k=1;

is\_game=false;

unstopable=false;

ball\_count=1;

ball\_pos=(struct TBallPosition \*)realloc(ball\_pos, ball\_count\*sizeof(struct TBallPosition));

x\_speed=(int \*)realloc(x\_speed, ball\_count\*sizeof(int));

y\_speed=(int \*)realloc(y\_speed, ball\_count\*sizeof(int));

angle=(float \*)realloc(angle, ball\_count\*sizeof(float));

ball\_pos[0].x=plt\_pos+plt\_length-7;

ball\_pos[0].y=plt\_lvl-15;

x\_speed[0]=5;

y\_speed[0]=5;

s\_ball.setPosition((int)ball\_pos[0].x, (int)ball\_pos[0].y);

is\_bonus=false;

bonus\_pos.x=0;

bonus\_pos.y=0;

bonus\_pos.x1=0;

bonus\_pos.x2=0;

bonus\_pos.y1=0;

bonus\_pos.y2=0;

bx\_speed=0;

by\_speed=2;

bangle=1;

cur\_ball=0;

gun.is\_gun=false;

gun.shoot\_count=0;

shoots=(struct TShootPosition \*)realloc(shoots, gun.shoot\_count\*sizeof(struct TShootPosition));

lifes--;

if (lifes==0)

{

for (int i = 0; i < HEIGHT\_MAP; i++)

for (int j = 0; j < WIDTH\_MAP; j++)

TileMap[i][j]=' ';

end\_game=true;

}

continue;

}

cur\_ball--;

//continue;

}

}

if ((move\_timer>game\_speed)&&(ball\_pos[cur\_ball].y<(window\_size.bot/3\*2)))

{

//std::cout<<block\_count<<"\n";

//top

block\_x=(int)((ball\_pos[cur\_ball].x+7)/45);

block\_y=(int)(ball\_pos[cur\_ball].y/30);

if ((block\_x<WIDTH\_MAP) && (block\_y<HEIGHT\_MAP))

{

if(TileMap[block\_y][block\_x]!=' ')

{

//std::cout<<"top!\n";

if (TileMap[block\_y][block\_x]!='r')

{

block\_count--;

score+=10\*score\_k;

}

else

if (!unstopable)

sound\_red.play();

if (TileMap[block\_y][block\_x]=='i')

{

sound\_invis.play();

}

if (TileMap[block\_y][block\_x]=='y')

{

sound\_glass.play();

}

if (TileMap[block\_y][block\_x]=='g')

{

sound\_bblock.play();

breaking\_block.count++;

bblocks=(struct TBreakingP \*)realloc(bblocks, breaking\_block.count \* sizeof(struct TBreakingP));

bblocks[breaking\_block.count-1].x=block\_x\*45;

bblocks[breaking\_block.count-1].y=block\_y\*30;

bblocks[breaking\_block.count-1].step=1;

bblocks[breaking\_block.count-1].step\_time=0;

}

TileMap[block\_y][block\_x]=NextBlock(TileMap[block\_y][block\_x], unstopable);

if ((!is\_bonus)&&(TileMap[block\_y][block\_x]!='r'))

{

is\_bonus=BonusMaking(&bonus\_pos, &bx\_speed, x\_speed[cur\_ball], &bangle, block\_x, block\_y, &bonus\_type);

if (is\_bonus)

{

sound\_bonus.play();

s\_bonus.setTextureRect(IntRect(bonus\_pos.x1, bonus\_pos.y1, bonus\_pos.x2, bonus\_pos.y2));

// std::cout<<bonus\_pos.x1<<"\n";

s\_bonus.setPosition(bonus\_pos.x, bonus\_pos.y);

}

}

if(!unstopable)

{

y\_speed[cur\_ball]=-y\_speed[cur\_ball];

ball\_pos[cur\_ball].y++;

}

}

}

//right

block\_x=(int)((ball\_pos[cur\_ball].x+15)/45);

block\_y=(int)((ball\_pos[cur\_ball].y+7)/30);

if ((block\_x<WIDTH\_MAP) && (block\_y<HEIGHT\_MAP))

{

if(TileMap[block\_y][block\_x]!=' ')

{

//std::cout<<"right!\n";

if (TileMap[block\_y][block\_x]!='r')

{

block\_count--;

score+=10\*score\_k;

}

else

if (!unstopable)

sound\_red.play();

if (TileMap[block\_y][block\_x]=='i')

{

sound\_invis.play();

}

if (TileMap[block\_y][block\_x]=='y')

{

sound\_glass.play();

}

if (TileMap[block\_y][block\_x]=='g')

{

sound\_bblock.play();

breaking\_block.count++;

bblocks=(struct TBreakingP \*)realloc(bblocks, breaking\_block.count \* sizeof(struct TBreakingP));

bblocks[breaking\_block.count-1].x=block\_x\*45;

bblocks[breaking\_block.count-1].y=block\_y\*30;

bblocks[breaking\_block.count-1].step=1;

bblocks[breaking\_block.count-1].step\_time=0;

}

TileMap[block\_y][block\_x]=NextBlock(TileMap[block\_y][block\_x], unstopable);

if ((!is\_bonus)&&(TileMap[block\_y][block\_x]!='r'))

{

is\_bonus=BonusMaking(&bonus\_pos, &bx\_speed, x\_speed[cur\_ball], &bangle, block\_x, block\_y, &bonus\_type);

if (is\_bonus)

{

sound\_bonus.play();

s\_bonus.setTextureRect(IntRect(bonus\_pos.x1, bonus\_pos.y1, bonus\_pos.x2, bonus\_pos.y2));

// std::cout<<bonus\_pos.x1<<"\n";

s\_bonus.setPosition(bonus\_pos.x, bonus\_pos.y);

}

}

if(!unstopable)

{

x\_speed[cur\_ball]=-x\_speed[cur\_ball];

ball\_pos[cur\_ball].x=block\_x\*45-17;

}

}

}

//left

block\_x=(int)((ball\_pos[cur\_ball].x)/45);

block\_y=(int)((ball\_pos[cur\_ball].y+7)/30);

if ((block\_x<WIDTH\_MAP) && (block\_y<HEIGHT\_MAP))

{

if(TileMap[block\_y][block\_x]!=' ')

{

// std::cout<<"left!\n";

if (TileMap[block\_y][block\_x]!='r')

{

block\_count--;

score+=10\*score\_k;

}

else

if (!unstopable)

sound\_red.play();

if (TileMap[block\_y][block\_x]=='i')

{

sound\_invis.play();

}

if (TileMap[block\_y][block\_x]=='y')

{

sound\_glass.play();

}

if (TileMap[block\_y][block\_x]=='g')

{

sound\_bblock.play();

breaking\_block.count++;

bblocks=(struct TBreakingP \*)realloc(bblocks, breaking\_block.count \* sizeof(struct TBreakingP));

bblocks[breaking\_block.count-1].x=block\_x\*45;

bblocks[breaking\_block.count-1].y=block\_y\*30;

bblocks[breaking\_block.count-1].step=1;

bblocks[breaking\_block.count-1].step\_time=0;

}

TileMap[block\_y][block\_x]=NextBlock(TileMap[block\_y][block\_x], unstopable);

if ((!is\_bonus)&&(TileMap[block\_y][block\_x]!='r'))

{

is\_bonus=BonusMaking(&bonus\_pos, &bx\_speed, x\_speed[cur\_ball], &bangle, block\_x, block\_y, &bonus\_type);

if (is\_bonus)

{

sound\_bonus.play();

s\_bonus.setTextureRect(IntRect(bonus\_pos.x1, bonus\_pos.y1, bonus\_pos.x2, bonus\_pos.y2));

// std::cout<<bonus\_pos.x1<<"\n";

s\_bonus.setPosition(bonus\_pos.x, bonus\_pos.y);

}

}

if(!unstopable)

{

x\_speed[cur\_ball]=-x\_speed[cur\_ball];

ball\_pos[cur\_ball].x=block\_x\*45+47;

}

}

}

//bot

block\_x=(int)((ball\_pos[cur\_ball].x+7)/45);

block\_y=(int)((ball\_pos[cur\_ball].y+15)/30);

if ((block\_x<WIDTH\_MAP) && (block\_y<HEIGHT\_MAP))

{

if(TileMap[block\_y][block\_x]!=' ')

{

//std::cout<<"bot!\n";

if (TileMap[block\_y][block\_x]!='r')

{

block\_count--;

score+=10\*score\_k;

}

else

if (!unstopable)

sound\_red.play();

if (TileMap[block\_y][block\_x]=='i')

{

sound\_invis.play();

}

if (TileMap[block\_y][block\_x]=='y')

{

sound\_glass.play();

}

if (TileMap[block\_y][block\_x]=='g')

{

sound\_bblock.play();

breaking\_block.count++;

bblocks=(struct TBreakingP \*)realloc(bblocks, breaking\_block.count \* sizeof(struct TBreakingP));

bblocks[breaking\_block.count-1].x=block\_x\*45;

bblocks[breaking\_block.count-1].y=block\_y\*30;

bblocks[breaking\_block.count-1].step=1;

bblocks[breaking\_block.count-1].step\_time=0;

}

TileMap[block\_y][block\_x]=NextBlock(TileMap[block\_y][block\_x], unstopable);

if ((!is\_bonus)&&(TileMap[block\_y][block\_x]!='r'))

{

is\_bonus=BonusMaking(&bonus\_pos, &bx\_speed, x\_speed[cur\_ball], &bangle, block\_x, block\_y, &bonus\_type);

if (is\_bonus)

{

sound\_bonus.play();

s\_bonus.setTextureRect(IntRect(bonus\_pos.x1, bonus\_pos.y1, bonus\_pos.x2, bonus\_pos.y2));

// std::cout<<bonus\_pos.x1<<"\n";

s\_bonus.setPosition(bonus\_pos.x, bonus\_pos.y);

}

}

if(!unstopable)

{

y\_speed[cur\_ball]=-y\_speed[cur\_ball];

ball\_pos[cur\_ball].y--;

}

}

}

}

if (ball\_pos[cur\_ball].y<=window\_size.top)

{

sound\_wall.play();

y\_speed[cur\_ball]=-y\_speed[cur\_ball];

ball\_pos[cur\_ball].y=window\_size.top+1;

}

if (ball\_pos[cur\_ball].x<=window\_size.left)

{

sound\_wall.play();

x\_speed[cur\_ball]=-x\_speed[cur\_ball];

ball\_pos[cur\_ball].x=window\_size.left+1;

}

if (ball\_pos[cur\_ball].x>=window\_size.right-15)

{

sound\_wall.play();

x\_speed[cur\_ball]=-x\_speed[cur\_ball];

ball\_pos[cur\_ball].x=window\_size.right-16;

}

if (is\_bonus)

{

if (bonus\_pos.y<=window\_size.top)

{

by\_speed\*=-1;

bonus\_pos.y=window\_size.top+1;

}

if (bonus\_pos.x<=window\_size.left)

{

bx\_speed\*=-1;

bonus\_pos.x=window\_size.left+1;

}

if (bonus\_pos.x>=window\_size.right-40)

{

bx\_speed\*=-1;

bonus\_pos.x=window\_size.right-40;

}

}

if (move\_timer>game\_speed)

{

//ball.sprite.setPosition((int)(ball\_pos[cur\_ball].x + x\_speed\*angle),(int)(ball\_pos[cur\_ball].y + y\_speed));

ball\_pos[cur\_ball].x=(ball\_pos[cur\_ball].x + x\_speed[cur\_ball]\*angle[cur\_ball]);

ball\_pos[cur\_ball].y=(ball\_pos[cur\_ball].y + y\_speed[cur\_ball]);

x\_disp=(ball\_pos[cur\_ball].x + x\_speed[cur\_ball]\*angle[cur\_ball])-ball\_pos[cur\_ball].x;

// std::cout<<"count="<<ball\_count<<"\n";

/\* std::cout<<"x="<<ball\_pos.x<<"\n";

std::cout<<"y="<<ball\_pos.y<<"\n";

std::cout<<"x\_speed="<<x\_speed<<"\n";

std::cout<<"y\_speed="<<y\_speed<<"\n";

std::cout<<"angle="<<angle<<"\n";

std::cout<<"x\_disp="<<x\_disp<<"\n";

// std::cout<<"time="<<time<<"\n";\*/

//move\_timer=0;

}

}

//BONUS\_MOVE

if (is\_bonus)

{

if (move\_timer>game\_speed)

{

bonus\_pos.x=bonus\_pos.x + bx\_speed \* bangle;

bonus\_pos.y=bonus\_pos.y + by\_speed;

if (bangle>0)

bangle-=0.05;

}

}

//SHOOT\_MOVE

if(gun.is\_gun)

{

if (move\_timer>game\_speed)

{

if(gun.shoot\_count>0)

for (int i\_shoot=0; i\_shoot< gun.shoot\_count; i\_shoot++)

{

shoots[i\_shoot].y+=gun.shoot\_speed;

if (shoots[i\_shoot].y>0)

{

block\_x=(int)((shoots[i\_shoot].x+1)/45);

block\_y=(int)(shoots[i\_shoot].y/30);

if ((block\_x<WIDTH\_MAP) && (block\_y<HEIGHT\_MAP))

{

if(TileMap[block\_y][block\_x]!=' ')

{

//std::cout<<"top!\n";

if (TileMap[block\_y][block\_x]!='r')

{

block\_count--;

score+=10\*score\_k;

}

if (TileMap[block\_y][block\_x]=='i')

{

sound\_invis.play();

}

if (TileMap[block\_y][block\_x]=='y')

{

sound\_glass.play();

}

if (TileMap[block\_y][block\_x]=='g')

{

sound\_bblock.play();

breaking\_block.count++;

bblocks=(struct TBreakingP \*)realloc(bblocks, breaking\_block.count \* sizeof(struct TBreakingP));

bblocks[breaking\_block.count-1].x=block\_x\*45;

bblocks[breaking\_block.count-1].y=block\_y\*30;

bblocks[breaking\_block.count-1].step=1;

bblocks[breaking\_block.count-1].step\_time=0;

}

TileMap[block\_y][block\_x]=NextBlock(TileMap[block\_y][block\_x], false);

// block\_count--;

if ((!is\_bonus)&&(TileMap[block\_y][block\_x]!='r'))

{

is\_bonus=BonusMaking(&bonus\_pos, &bx\_speed, 5, &bangle, block\_x, block\_y, &bonus\_type);

if (is\_bonus)

{

sound\_bonus.play();

s\_bonus.setTextureRect(IntRect(bonus\_pos.x1, bonus\_pos.y1, bonus\_pos.x2, bonus\_pos.y2));

// std::cout<<bonus\_pos.x1<<"\n";

s\_bonus.setPosition(bonus\_pos.x, bonus\_pos.y);

}

}

if (i\_shoot!=gun.shoot\_count-1)

{

//std::cout<<"shift\n";

for (int i=i\_shoot; i<gun.shoot\_count-1; i++)

{

shoots[i]=shoots[i+1];

}

}

gun.shoot\_count--;

shoots=(struct TShootPosition \*)realloc(shoots, gun.shoot\_count\*sizeof(struct TShootPosition));

}

}

}

else

{

if (i\_shoot!=gun.shoot\_count-1)

{

//std::cout<<"shift\n";

for (int i=i\_shoot; i<gun.shoot\_count-1; i++)

{

shoots[i]=shoots[i+1];

}

}

gun.shoot\_count--;

shoots=(struct TShootPosition \*)realloc(shoots, gun.shoot\_count\*sizeof(struct TShootPosition));

}

}

}

}

}

if (move\_timer>game\_speed)

move\_timer=0;

window.clear();

//std::cout<<block\_count<<"\n";

//NEW LVL

if(IsEnd()&&!end\_game)

{

GenerateMap();

plt\_length=100;

s\_plt.setTextureRect(IntRect(0, GetPlatformY(plt\_length), plt\_length, 19));

score\_k=1;

gun.is\_gun=false;

sound\_fanfare.play();

is\_game=false;

unstopable=false;

gun.shoot\_count=0;

shoots=(struct TShootPosition \*)realloc(shoots, gun.shoot\_count\*sizeof(struct TShootPosition));

for (int i=1; i<ball\_count-1; i++)

{

ball\_pos[i]=ball\_pos[i+1];

x\_speed[i]=x\_speed[i+1];

y\_speed[i]=y\_speed[i+1];

angle[i]=angle[i+1];

}

ball\_count=1;

ball\_pos=(struct TBallPosition \*)realloc(ball\_pos, ball\_count\*sizeof(struct TBallPosition));

x\_speed=(int \*)realloc(x\_speed, ball\_count\*sizeof(int));

y\_speed=(int \*)realloc(y\_speed, ball\_count\*sizeof(int));

x\_speed[0]=5;

y\_speed[0]=5;

angle=(float \*)realloc(angle, ball\_count\*sizeof(float));

bonus\_pos.x=0;

bonus\_pos.y=0;

bonus\_pos.x1=0;

bonus\_pos.x2=0;

bonus\_pos.y1=0;

bonus\_pos.y2=0;

bx\_speed=0;

by\_speed=2;

bangle=1;

is\_bonus=false;

s\_ball.setPosition(plt\_length/2-7, plt\_lvl-15);

}

//ПРОРИСОВКА

//КАРТА

for (int i = 0; i < HEIGHT\_MAP; i++)

for (int j = 0; j < WIDTH\_MAP; j++)

{

//if (TileMap[i][j] == '0') s\_map.setTextureRect(IntRect(0, 0, 30, 20)); //если встретили символ пробел, то рисуем 1й квадратик

//if (TileMap[i][j] == ' ') s\_map.setTextureRect(IntRect(33, 0, 30, 20));//если встретили символ s, то рисуем 2й квадратик

if ((TileMap[i][j] == 'g')) s\_map.setTextureRect(IntRect(0, 0, 45, 30));

if ((TileMap[i][j] == 'y')) s\_map.setTextureRect(IntRect(46, 0, 45, 30));

if ((TileMap[i][j] == 'i')) s\_map.setTextureRect(IntRect(0, 0, 0, 0));

if ((TileMap[i][j] == 'r')) s\_map.setTextureRect(IntRect(92, 0, 45, 30));

if ((TileMap[i][j] == ' ')) s\_map.setTextureRect(IntRect(0, 0, 0, 0));

s\_map.setPosition(j \* 45, i \* 30);//по сути раскидывает квадратики, превращая в карту. то есть задает каждому из них позицию. если убрать, то вся карта нарисуется в одном квадрате 32\*32 и мы увидим один квадрат

window.draw(s\_map);//рисуем квадратики на экран

}

//Платформа

window.draw(s\_plt);

//Разрушающиеся блоки

for (int i=0; i<breaking\_block.count; i++)

{

if (bblocks[i].step<=2)

{

s\_breaking.setTextureRect(IntRect((bblocks[i].step-1)\*46, 30, 45, 30));

s\_breaking.setPosition(bblocks[i].x, bblocks[i].y);

window.draw(s\_breaking);

}

else

{

if (i!=breaking\_block.count-1)

{

//std::cout<<"shift\n";

for (int j=i; j<breaking\_block.count-1; j++)

{

bblocks[j]=bblocks[j+1];

}

}

breaking\_block.count--;

bblocks=(struct TBreakingP \*)realloc(bblocks, breaking\_block.count \* sizeof(struct TBreakingP));

}

}

//Выстрелы

if (gun.is\_gun)

{

s\_gun.setTextureRect(IntRect(0,0,10,20));

s\_gun.setPosition(plt\_pos+10, plt\_lvl-15);

window.draw(s\_gun);

s\_gun.setPosition(plt\_pos+plt\_length-20, plt\_lvl-15);

window.draw(s\_gun);

if (gun.shoot\_count>0)

{

s\_gun.setTextureRect(IntRect(10,0,4,5));

for (int i=0; i<gun.shoot\_count; i++)

{

s\_gun.setPosition(shoots[i].x, shoots[i].y);

window.draw(s\_gun);

}

}

}

//Мячи

for (cur\_ball=0; cur\_ball<ball\_count; cur\_ball++)

{

s\_ball.setPosition((int)(ball\_pos[cur\_ball].x),(int)(ball\_pos[cur\_ball].y));

window.draw(s\_ball);

}

//Бонус

if (is\_bonus)

{

s\_bonus.setPosition(bonus\_pos.x, bonus\_pos.y);

window.draw(s\_bonus);

}

//Текст

std::ostringstream score\_string;

score\_string << score;

text.setString("Счет: "+score\_string.str());

text.setPosition(window\_size.left+10, window\_size.bot-30);

window.draw(text);

if (end\_game)

{

text.setString("Конец игры! Вы набрали "+score\_string.str()+" очков");

text.setPosition(window\_size.right/2-200, window\_size.bot/2-10);

window.draw(text);

}

std::ostringstream life\_string;

life\_string << lifes;

text.setString("Жизней: "+life\_string.str());

text.setPosition(window\_size.right-150, window\_size.bot-30);

window.draw(text);

text.setString("Включить/выключить паузу - Пробел");

text.setPosition(window\_size.right/2-100, window\_size.bot-30);

window.draw(text);

if (cursor\_mess)

{

text.setString("Курсор за пределами окна!");

text.setPosition(window\_size.right/2-200, window\_size.bot-250);

window.draw(text);

}

window.display();

}

if (is\_pause)

{

text.setString("Пауза");

text.setPosition(window\_size.right/2-30, window\_size.bot-250);

window.draw(text);

window.display();

}

}

return 0;

}

Map.h

#include <SFML\Graphics.hpp>

#include <time.h>

const int HEIGHT\_MAP = 20;//размер карты высота

const int WIDTH\_MAP = 26;//размер карты ширина

sf::String TileMap[HEIGHT\_MAP];

int block\_count;

void CheckingMap()

{

int i, j;

for (i = 1; i < HEIGHT\_MAP-1; i++)

for (j = 1; j < WIDTH\_MAP-1; j++)

if((TileMap[i-1][j]=='r')&&(TileMap[i][j-1]=='r')&&(TileMap[i][j+1]=='r')&&(TileMap[i+1][j]=='r'))

TileMap[i-1][j]=' ';

i=0;

for (j=1; j < WIDTH\_MAP-1; j++)

if((TileMap[i+1][j]=='r')&&(TileMap[i][j-1]=='r')&&(TileMap[i][j+1]=='r'))

TileMap[i+1][j]=' ';

j=0;

for (i = 1; i < HEIGHT\_MAP-1; i++)

if((TileMap[i-1][j]=='r')&&(TileMap[i+1][j]=='r')&&(TileMap[i][j+1]=='r'))

TileMap[i-1][j]=' ';

j=WIDTH\_MAP;

for (i = 1; i < HEIGHT\_MAP-1; i++)

if((TileMap[i-1][j]=='r')&&(TileMap[i+1][j]=='r')&&(TileMap[i][j-1]=='r'))

TileMap[i-1][j]=' ';

if((TileMap[0][1]=='r')&&(TileMap[1][0]=='r'))

TileMap[1][0]=' ';

if((TileMap[0][WIDTH\_MAP-1]=='r')&&(TileMap[1][WIDTH\_MAP]=='r'))

TileMap[1][WIDTH\_MAP]=' ';

}

void GenerateMap()

{

bool x;

block\_count=0;

int amount, r, type, t;

char s[WIDTH\_MAP], type\_ch;

srand(unsigned(time(NULL)));

amount=rand()%10+1;

for (int i=0; i<20; i++)

{

for (int j=0; j<13; j++)

{

r=rand()%10;

if (r<amount)

{

x=1;

type=rand()%4+1;

switch (type)

{

case 1: type\_ch='g'; t=1; break;

case 2: type\_ch='y'; t=2; break;

case 3: type\_ch='i'; t=2; break;

case 4: type\_ch='r'; t=0; break;

}

}

else

x=0;

if (x)

{

block\_count+=2\*t;

s[j]=type\_ch;

s[WIDTH\_MAP-j-1]=type\_ch;

}

else

{

s[j]=' ';

s[WIDTH\_MAP-j-1]=' ';

}

}

TileMap[i]=s+'\0';

}

CheckingMap();

// std::cout<<block\_count<<"\n";

}