Комитет по образованию г. Санкт-Петербург

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ПРЕЗИДЕНТСКИЙ ФИЗИКО-МАТЕМАТИЧЕСКИЙ
ЛИЦЕЙ №239

Отчет о практике
«Создание графических приложений на языке C++»

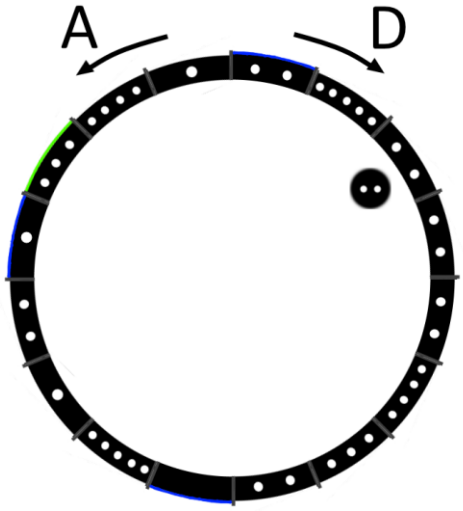
Учащийся 10-1 класса
Никитин Ф.С.

Преподаватель:
Клюнин А.О.

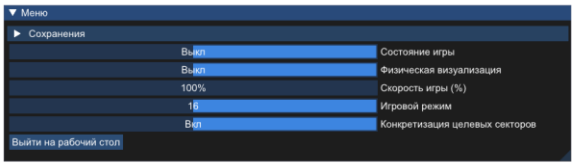
Санкт-Петербург – 2022 год

1. Постановка задачи

Написание простейшей аркадной игры с несколькими режимами.



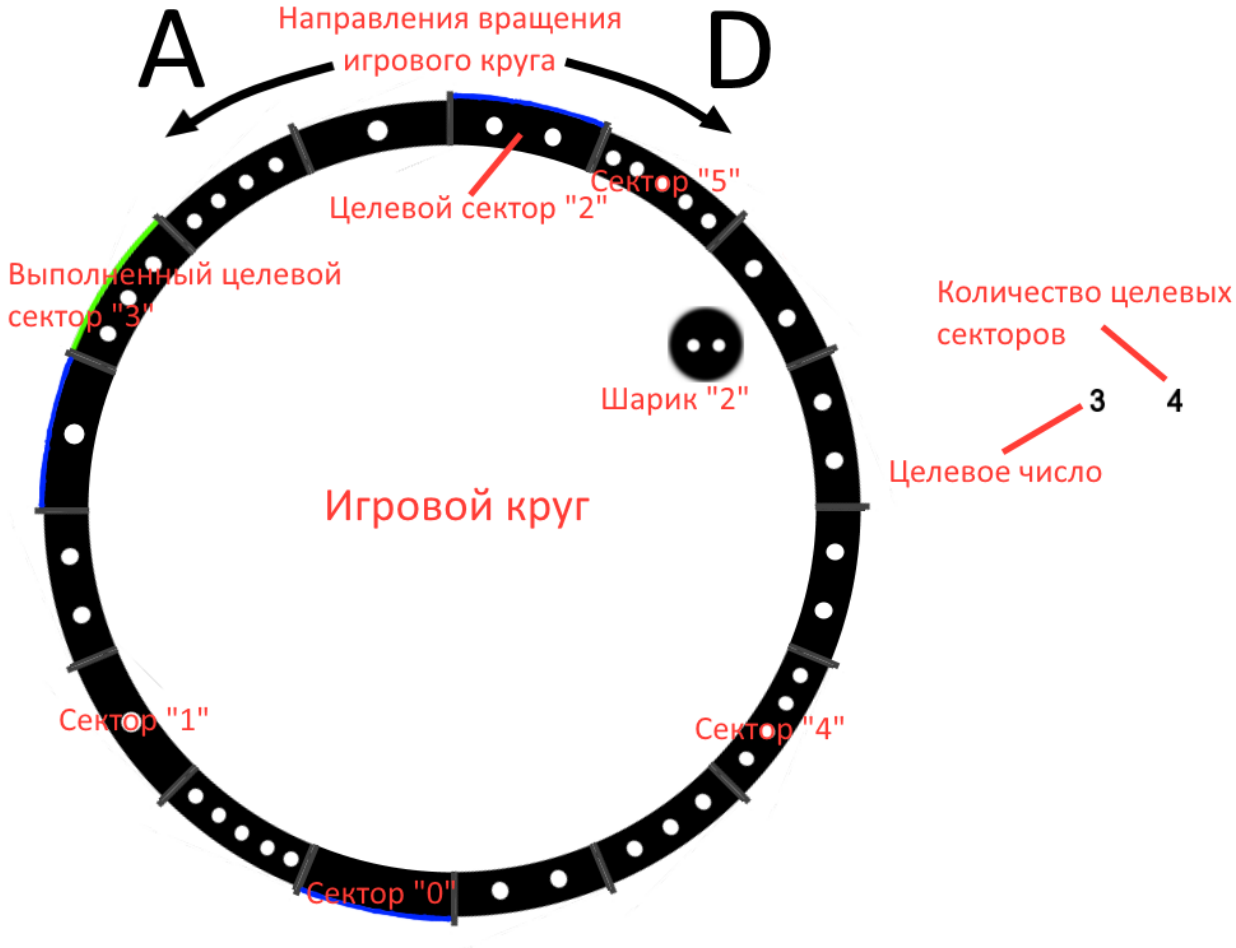
3 4



2. Элементы управления

Игра включается, а меню закрывается с помощью кнопки «Состояние игры». Для проверки работы физической механики игры и упрощения игры может включаться «Физическая визуализация». По просьбам добровольных тестировщиков добавлено изменение скорости игры. Кнопка «Игровой режим» меняет размер игрового поля между 12 и 16. Кнопка «Конкретизация целевых секторов» добавляет упрощенный режим игры с неконкретизированными целевыми секторами. Кнопка «Выйти на рабочий стол» закрывает игру.

3. Игровой процесс



Игровой процесс происходит в игровом круге, разделенным на несколько(12,16) секторов, в каждом из которых написан остаток по модулю 6. Игровой круг вращается и вращения можно ускорять в ту или иную сторону с помощью клавиш A/D. Внутри игрового круга катается шарик, время от времени соударяясь со стенками. При этом числа в шарике и секторе соударения суммируются по модулю 6, после чего записываются в оба объекта.

Синяя каемка выделяет среди секторов целевые. Цель игры - помещение в них целевого числа, выводимого справа от игрового круга. При достижении этого для конкретного сектора каемка меняет цвет с синего на зеленый. По достижению этого для всех секторов игра завершается победой игрока. Проигрышем же игра завершается в случае поломки шарика от нагрузок, наступающих вследствие частых соприкосновений его с поверхностью игрового круга. В режиме неконкретизированных целевых секторов все сектора целевые, но для победы целевое значения должно быть в целевом их количестве, написанном правее целевого значения. В обычном режиме там же пишется количество целевых секторов.

4. Структуры данных

Взаимодействия игрового круга и его секторов выведены в класс Circle.

```
#ifndef OTSKOK_CIRCLE_H
#define OTSKOK_CIRCLE_H


#include <SFML/Graphics.hpp>
#include "cmath"
#include "iostream"
#include "thread"

using namespace sf;

struct Segment
{
public:
    Segment();

    Sprite getSprite();

    void setGameTargetCondition(int game_target_condition);

    void setPosition(int type, double angle, Vector2f position, Texture &texture);

    void update(double angle);
    void update(int condition);
    void update(bool target);

    void checkExecution(int tar);

    bool isExecution();

    int getCondition();

    double getAngle();

private:
    Sprite sprite;

    Texture texture;

    int type;

    const int width[2] = { [0] 90, [1] 71};
    const int height[2] = { [0] 190, [1] 142};
    const Vector2f texturePosition[2] = { [0] Vector2f( x: 426, y: 0), [1] Vector2f( x: 0, y: 0)};
    const Vector2f spriteOrigin[2] = { [0] Vector2f( x: -234.0, y: 10.0), [1] Vector2f( x: -260.0, y: 10.0)};

    int condition;
    bool target;
    bool game_target_condition;
    bool is_execution;

    double angle;
};
```

```cpp
class Circle
{
public:
    Circle();

    void update(float elapsedTime);

    double w_speed;

    int width = 40;

    int radius;

    Vector2f centerPosition;

    Segment round[16];

    Texture segment;

    int type;
    int size[2] = { 0: 12, 1: 16};
};


#endif //OTSKOK_CIRCLE_H
```

```cpp
#include "Circle.h"

Segment::Segment()
{
    condition = 0;
    target = true;

    angle = 0.0;
}

Sprite Segment::getSprite()
{
    return sprite;
}

void Segment::setPosition(int type, double angle, Vector2f position, Texture &texture)
{
    this->type = type;
    sprite.rotate( angle: -this->angle);
    this->angle = angle;
    sprite.setTexture(texture);
    int texture_line = 0;
    if(game_target_condition)
    {
        texture_line = target;
    }
    if(is_execution)
    {
        texture_line = 3;
    }
    sprite.setTextureRect( rectangle: IntRect( rectLeft: texturePosition[type].x+width[type]*condition, rectTop: texturePosition[type].y+height[type]*texture_line, rectWidth: width[type], rectHeight: height[type]));
    sprite.setOrigin(spriteOrigin[type]);
    sprite.rotate(angle);
    sprite.setPosition(position);
}

void Segment::update(double angle)
{
    this->angle += angle;
    sprite.rotate(angle);
}

void Segment::update(int condition)
{
    this->condition = condition;
    int texture_line = 0;
    if(game_target_condition)
    {
        texture_line = target;
    }
    if(is_execution)
    {
        texture_line = 2;
    }
    sprite.setTextureRect( rectangle: IntRect( rectLeft: texturePosition[type].x+width[type]*condition, rectTop: texturePosition[type].y+height[type]*texture_line, rectWidth: width[type], rectHeight: height[type]));
```

```
    sprite.setTextureRect( rectangle: IntRect( rectLeft: texturePosition[type].x+width[type]*condition, rectTop: texturePosition[type].y+height[type]*texture_line, rectWidth: width[type], rectHeight: height[type]));
}

void Segment::update(bool target) {
    this->target = target;
}

void Segment::checkExecution(int tar) {
    if(target != 1 || condition != tar)
    {
        is_execution = false;
    }
    else{
        is_execution =  true;
    }
}

int Segment::getCondition() {
    return condition;
}

double Segment::getAngle() {
    return angle;
}

void Segment::setGameTargetCondition(int game_target_condition) {
    this->game_target_condition = game_target_condition;
}

bool Segment::isExecution() {
    return is_execution;
}


Circle::Circle()
{
    w_speed = 0.0;

    centerPosition.x = 400;
    centerPosition.y = 400;

    segment.loadFromFile( filename: "resources/segment.png");
    segment.setSmooth(true);

    radius = 324;
}

void Circle::update(float elapsedTime)
{
    for(int i = 0;i < size[type]; i++)
    {
        round[i].update( angle: elapsedTime*w_speed);
    }
}
```

Взаимодействия шарика выведены в класс Ball

```
#ifndef OTSKOK_BALL_H
#define OTSKOK_BALL_H
#include <SFML/Graphics.hpp>
#include "Circle.h"

struct event_impact
{
    Vector2f center_position;
    Vector2f impact_position;
    std::pair<Vector2f, Vector2f> speed;
    double round_angle;
    double angle;
    int ans;
    double time_after_impact;
    event_impact(){
        center_position = Vector2f ( X: 0, Y: 0);
        impact_position = Vector2f ( X: 0, Y: 0);
        speed.first = Vector2f ( X: 0, Y: 0);
        speed.second = Vector2f ( X: 0, Y: 0);
        round_angle = 0.0;
        angle = 0.0;
        ans = 0;
        time_after_impact = 0.0;
    }
};

using namespace sf;

class Ball
{
private:
    Sprite sprite;
    Texture* texture;

public:
    event_impact last_impact;

    Vector2f position;

    Vector2f speed;
    float w_speed;
    float k_energy;
    Ball();
    Sprite getSprite();
    void setSprite(Texture &texture);
    int radius = 30;
    int condition;
    bool checkPosition(Circle circle);
    int checkSegment(Circle circle);
    void update(float elapsedTime);
    void update(int conidition);
};
#endif //OTSKOK_BALL_H
```

```cpp
#include "Ball.h"
#include "Function.h"



Ball::Ball()
{
    speed.x = 70;
    speed.y = 0;
    w_speed = 0.0;
    k_energy = speed.x*speed.x + speed.y*speed.y;

    position.x = 300;
    position.y = 250;

    condition = 0;

}

Sprite Ball::getSprite()
{
    return sprite;
}

void Ball::setSprite(Texture &texture)
{
    sprite.setTexture(texture);

    sprite.setTextureRect( rectangle: IntRect( rectLeft: 2*condition*radius, rectTop: 0, rectWidth: 2*radius, rectHeight: 2*radius));

    sprite.setOrigin( x: radius,  y: radius);
}

void Ball::update(int condition)
{
    this->condition = condition;
    sprite.setTextureRect( rectangle: IntRect( rectLeft: 2*condition*radius, rectTop: 0, rectWidth: 2*radius, rectHeight: 2*radius));
}

bool Ball::checkPosition(Circle circle)
{
    long double r = (position.x-circle.centerPosition.x)*(position.x-circle.centerPosition.x) + (position.y-circle.centerPosition.y)*(position.y-circle.centerPosition.y);
    if(((circle.radius - circle.width - radius)*(circle.radius - circle.width - radius) - r) < 0)
    {
        return true;
    }
    return false;
}
int Ball::checkSegment(Circle circle)
{
    Vector2f delta = position - circle.centerPosition;
    double angle = 180/M_PI*acos( x: delta.x/sqrt( x: delta.x*delta.x + delta.y*delta.y));
    if(position.y < circle.centerPosition.y)
    {
        angle = 360 - angle;
    }
    last_impact.impact_position = Vector2f ( x: circle.radius - circle.width, y: 0);
    last_impact.impact_position = rotation( z: last_impact.impact_position,  angle: angle*M_PI/180);
    if(abs( x: last_impact.angle - angle) < 10.0)
    {
        return -1;
    }
    last_impact.angle = angle;
    angle -= circle.round[0].getAngle();
    last_impact.round_angle = circle.round[0].getAngle();
    while(angle < 0)
    {
        angle += 360;
    }
    int ans = angle/360.0*circle.size[circle.type];
    ans = ans % circle.size[circle.type];
    last_impact.ans = ans;
    return ans;
}

void Ball::update(float elapsedTime)
{
    last_impact.time_after_impact += elapsedTime;
    sprite.rotate( angle: w_speed*elapsedTime);
    position += speed*elapsedTime;
    sprite.setPosition(position);

}
```

Часто используемые функции выведены в класс Function

```cpp
#ifndef OTSKOK_FUNCTION_H
#define OTSKOK_FUNCTION_H

#include <imgui-SFML.h>
#include <imgui.h>
#include <SFML/Graphics.hpp>
#include "cmath"
#include <iostream>

using namespace sf;


Vector2f rotation(Vector2f a, double angle);

float scalar(Vector2f a, Vector2f b);

float max(float a, float b);

#endif //OTSKOK_FUNCTION_H
```

```cpp
#include "function.h"



Vector2f rotation(Vector2f a, double angle)
{
    Vector2f ans;
    ans.x = a.x*cos( x: angle) - a.y*sin( x: angle);
    ans.y = a.x*sin( x: angle) + a.y*cos( x: angle);
    return ans;
}
float scalar(Vector2f a, Vector2f b)
{
    return a.x*b.x + a.y*b.y;
}

float max(float a, float b)
{
    if(a > b)
        return a;
    return b;
}
```

Ввод выведен в Input.cpp

```cpp
#include "Engine.h"

void Engine::input()
{
    if (Keyboard::isKeyPressed(Keyboard::A))
    {
        m_Circle.w_speed -= 0.25;
    }

    if (Keyboard::isKeyPressed(Keyboard::D))
    {
        m_Circle.w_speed += 0.25;
    }
    if(Keyboard::isKeyPressed(Keyboard::Escape))
    {
        condition_game = game_Close;
    }
}
```

Игра запускается main.cpp

```cpp
#include "Engine.h"
#include "iostream"
#include "cmath"
#include "iomanip"

int main()
{
    std::cout << std::setprecision( n: 10);
    srand( Seed: time( Time: 0 ));
    Engine engine;
    engine.start();
    return 0;
}
```

Вывод выведен в Draw.cpp

```cpp
#include "Engine.h"
#include "Function.h"
#include "iostream"
#include "thread"
void Engine::impact_draw()
{
    // задаём левый верхний край невидимого окна
    ImGui::SetNextWindowPos( pos: ImVec2( x: 0,  y: 0));
    // задаём правый нижний край невидимого окна
    ImGui::SetNextWindowSize( size: Resolution);
    // создаём невидимое окно
    ImGui::Begin( name: "Last_Impact",  p_open: nullptr,
                  flags: ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove |
                 ImGuiWindowFlags_NoScrollbar | ImGuiWindowFlags_NoInputs | ImGuiWindowFlags_NoBackground);
    // получаем список примитивов, которые будут нарисованы
    auto pDrawList :ImDrawList* = ImGui::GetWindowDrawList();

    pDrawList->AddLine(
             p1: m_Ball.last_impact.center_position,
             p2: m_Ball.last_impact.center_position + m_Ball.last_impact.speed.first,
             col: ImColor( r: 200, g: 100, b: 100),
             thickness: 5
            );
    pDrawList->AddLine(
             p1: m_Ball.last_impact.center_position,
             p2: m_Ball.last_impact.center_position + m_Ball.last_impact.speed.second,
             col: ImColor( r: 100, g: 200, b: 100),
             thickness: 5
    );
    pDrawList->AddCircleFilled(
             center: m_Circle.centerPosition + m_Ball.last_impact.impact_position,
             radius: 10,
             col: ImColor( r: 100, g: 100, b: 200)
            );
    ImGui::End();
}

void Engine::draw()
{
    m_Window.clear( color: Color::White);

    if(condition_physicOverlay == physicOverlay_Open)
    {
        impact_draw();
    }

    m_Window.draw( drawable: m_BackgroundSprite);
    m_Window.draw( drawable: m_Ball.getSprite());
    for(int i = 0; i < m_Circle.size[m_Circle.type]; i++)
    {
        m_Window.draw( drawable: m_Circle.round[i].getSprite());
    }
    m_Window.draw( drawable: target_text);
    m_Window.draw( drawable: target_num_text);
```

```
    m_Window.draw( drawable: target_text);
    m_Window.draw( drawable: target_num_text);
    ImGui::SFML::Render( &: m_Window);
    m_Window.display();
}

void Engine::endEvent(int type_of_end)
{
    m_Window.clear( color: Color::White);
    Text text;
    std::wstring s = L"";
    switch(type_of_end){
case 1:
    s += L"Шарик сломался из-за продолжительного соприкосновения с поверхностью. ";
    break;
case 2:
     s += L"ВЫ ПОБЕДИЛИ!!! ";
    break;
    }
    s += L"Для выхода из игры нажмите Esc";
    Vector2f position;
    position.x = 300;
    position.y = 0;
    text.setFont(font);
    text.setCharacterSize( size: 24);
    text.setString(s);
    text.getString();
    text.setFillColor( color: Color::Black);
    text.setStyle(sf::Text::Bold);
    m_Window.draw( drawable: text);
    m_Window.display();
    while(!Keyboard::isKeyPressed( key: Keyboard::Escape))
    {
        std::this_thread::sleep_for( rtime: std::chrono::milliseconds( rep: 10));
    }
    m_Window.close();
}

void Engine::event(int event_id)
{
    //сразу завершаем функцию, если ничего выводить не нужно
    if(event_id == 0)
    {
        return;
    }

    //задаем ширину выводимого текста
    const float wrap_width = 200.0;
    //задаем верхний-левый угол окна события
    Vector2f position = Resolution*0.5f;

    //выбираем текст в зависимости от типа события
    std::string text;
    switch(event_id){
```

```
    switch(event_id){
        case 1:
            text = u8"Шарик сломался из-за продолжительного соприкосновения с поверхностью";
            break;
        case 2:
            text = u8"ВЫ ПОБЕДИЛИ!!!";
            break;
        case 3:
            text = u8"Установите скорость игры в меню на 100%";
            break;
        case 4:
            text = u8"Включите физическую визуализацию, сдвинув соответственный тумблер в меню. Она позволит вам на первых порах лучше понимать логику отскоков";
            break;
        case 5:
            text = u8"Снимите игру с паузы, сдвинув соответственный тумблер в меню";
            break;
        case 6:
            text = u8"Только что произошло самое главное событие в игре - столкновение шарика с вращающимся ободком. В результате направление и скорость движения "
                   "шарика изменились(физическая визуализация показывает вам как) и числа, записанные в секторе, с которым столкнулся шарик, и в нем самом изменились "
                   "и стали равны сумме изначальных по модулю 6. Число, стоящее в шарике или секторе визуализируется количеством точек в нем и принимает значения от 0 до 5. "
                   "Для продолжения игры и закрытия окна сдвиньте тумблер в меню. ";
            break;
        case 7:
            text = u8"";
            break;
    }

    // задаём левый верхний край невидимого окна
    ImGui::SetNextWindowPos( pos: position);
    // задаём правый нижний край невидимого окна
    ImGui::SetNextWindowSize( size: event_resolution+Vector2f( X: 10, Y: 10));
    //создаем окно
    ImGui::Begin( name: "text",  p_open: nullptr,
                  flags: ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove |
                 ImGuiWindowFlags_NoScrollbar | ImGuiWindowFlags_NoInputs );

    //задаем ограничение на ширину текста
    ImGui::PushTextWrapPos( wrap_local_pos_x: ImGui::GetCursorPos().x + wrap_width);
    //пишем текст
    ImGui::Text( fmt: text.c_str(),wrap_width);

    //перерасчет ширины окна, чтобы в нее влезал текст
    event_resolution = ImGui::GetItemRectMax();
    event_resolution -= position;

    //рисуем рамочку вокруг текста
    auto draw_list :ImDrawList* = ImGui::GetWindowDrawList();
    draw_list->AddRect( p_min: ImGui::GetItemRectMin(),  p_max: event_resolution + position,  col: IM_COL32(255, 255, 0, 255));
    //отменяем ограничение на ширине текста
    ImGui::PopTextWrapPos();

    //закрываем окно
    ImGui::End();
}
```

Обновления состояний выведены в Update.cpp

```cpp
#include "Engine.h"
#include "Function.h"

static const char OUTPUT_PATH[255] = "resources/out.txt";

using namespace sf;

const double friction = 0.5;

// запись в файл
void Engine::saveToFile() {
    // открываем поток данных для записи в файл
    std::ofstream output( s: OUTPUT_PATH);

    // выводим в несколько строк информацию о предыдущем столкновению
    output << "||||||||||||||||||||||||||||||||||||||||||||||||||" << std::endl;
    output << m_Ball.last_impact.center_position.x << " " << m_Ball.last_impact.center_position.y <<  " Вектор на центр" << std::endl;
    output << m_Ball.last_impact.impact_position.x << " " << m_Ball.last_impact.impact_position.y << " Вектор на точку столкновения" << std::endl;
    output << m_Ball.last_impact.speed.first.x << " " << m_Ball.last_impact.speed.first.y << " Начальная скорость" << std::endl;
    output << m_Ball.last_impact.speed.second.x << " " << m_Ball.last_impact.speed.second.y << " Конечная скорость" << std::endl;
    output << m_Ball.last_impact.angle << " Угол на точку столкновения" << std::endl;
    output << m_Ball.last_impact.round_angle << " Угол проворота кольца" << std::endl;
    output << m_Ball.last_impact.ans << " Ожидаемый номер сектора попадания" << std::endl << std::endl;
    output << m_Ball.last_impact.time_after_impact << " Время, прошедшее с предыдущего столкновения" << std::endl << std::endl;
    // закрываем
    output.close();
}

// работа с файлами
void Engine::ShowFiles() {
    // если не раскрыта панель `Files`
    if (!ImGui::CollapsingHeader( label: "Сохранения"))
        // заканчиваем выполнение
        return;

    ImGui::PushID( int_id: 0);
    // создаём кнопку сохранения
    if (ImGui::Button( label: "Сохранить игру")) {
        // сохраняем задачу в файл
        saveToFile();
    }
    // восстанавливаем буфер id
    ImGui::PopID();

}

// если во всех целевых секторах стоит требуемое число игра завершается победой
bool Engine::checkWin()
{
    int ans = 0;
    for(int i = 0; i < m_Circle.size[m_Circle.type]; i++)
    {
        ans += m_Circle.round[i].isExecution();
    }
    return (ans == target_num);
```

```
        ans += m_Circle.round[i].isExecution();
    }
    return (ans == target_num);
}

void impact(Ball &m_Ball, Circle &m_Circle, float dtAsSeconds)
{
    m_Ball.speed = m_Ball.speed*m_Ball.k_energy/(m_Ball.k_energy - m_Ball.w_speed*m_Ball.w_speed*m_Ball.radius*2/5);
    if(dtAsSeconds > friction)
    {
        m_Ball.w_speed = m_Circle.w_speed;
    }
    else{
        m_Ball.w_speed += (m_Circle.w_speed - m_Ball.w_speed)*dtAsSeconds/friction;
    }
    m_Ball.speed = m_Ball.speed/m_Ball.k_energy*(m_Ball.k_energy - m_Ball.w_speed*m_Ball.w_speed*m_Ball.radius*2/5);
    Vector2f d = m_Ball.position - m_Circle.centerPosition;
    d = rotation( a: d,  angle: -m_Circle.w_speed*dtAsSeconds/2);
    float d_l = sqrt( x: d.x*d.x + d.y*d.y);
    d *= 2*scalar( a: d/d_l, b: m_Ball.speed)/d_l;
    m_Ball.speed = m_Ball.speed - d;
    m_Ball.last_impact.time_after_impact = 0;
}

void Engine::update(float dtAsSeconds)
{
    m_Ball.update( elapsedTime: dtAsSeconds);
    m_Circle.update( elapsedTime: dtAsSeconds);
    m_Ball.last_impact.impact_position = rotation( a: m_Ball.last_impact.impact_position, angle: dtAsSeconds*m_Circle.w_speed*M_PI/180);
    if(m_Ball.checkPosition( circle: m_Circle) && m_Ball.last_impact.time_after_impact > 0.1)
    {
        m_Ball.last_impact.center_position = m_Ball.position;
        m_Ball.last_impact.speed.first = m_Ball.speed;
        int i = m_Ball.checkSegment( circle: m_Circle);
        if(i == -1)
        {
            endEvent( type_of_damage: 1);
        }
        int ans = (m_Ball.condition + m_Circle.round[i].getCondition()) % 6;
        m_Circle.round[i].update( condition: ans);
        m_Circle.round[i].checkExecution( tar: target);
        m_Circle.round[i].update( condition: ans);
        m_Ball.update( conidition: ans);
        impact( &: m_Ball,  &: m_Circle, dtAsSeconds);
        m_Ball.last_impact.speed.second = m_Ball.speed;
        if(checkWin())
        {
            endEvent( type_of_damage: 2);
        }
    }
}
```

Игровой цикл расположен в классе Engine

```
#ifndef OTSKOK_ENGINE_H
#define OTSKOK_ENGINE_H

#include ...

using namespace sf;


class Engine
{
private:
    Clock clock;
    Vector2f Resolution;
    Vector2f event_resolution;
    RenderWindow m_Window;
    Sprite m_BackgroundSprite;
    Texture m_BackgroundTexture;
    Texture ball;
    Font font;

    Ball m_Ball;
    Circle m_Circle;

    int target;
    int target_num = 0;
    Text target_num_text;
    Text target_text;

    int game_speed = 100;

    enum physicOverlay { physicOverlay_Open, physicOverlay_Close };
    const char* physicOverlay_names[2] = { [0] "Вкл", [1] "Выкл"};
    int condition_physicOverlay = physicOverlay_Close;

    enum game { game_Open, game_Close };
    const char* game_names[2] = { [0] "Вкл", [1] "Выкл"};
    int condition_game = game_Close;

    enum mode { game_12, game_16 };
    const char* game_mode_names[2] = { [0] "12", [1] "16"};
    int  game_mode = game_16;

    enum game { game_Open, game_Close };
    const char* game_names[2] = { [0] "Вкл", [1] "Выкл"};
    int condition_game = game_Close;

    enum mode { game_12, game_16 };
    const char* game_mode_names[2] = { [0] "12", [1] "16"};
    int  game_mode = game_16;

    enum target_condition { no,  yes};
    const char* target_condition_names[2] = { [0] "Выкл", [1] "Вкл"};
    int  game_target_condition = yes;

    int menu();
    void setLevel();


    bool checkWin();

    void input();
    void update(float dtAsSeconds);
    void draw();
    void impact_draw();
    void endEvent(int type_of_damage);

    void saveToFile();
    void ShowFiles();

    void event(int event_id);
    int event_id = 0;

public:

    Engine();
    void start();
    void studing();
};


#endif //OTSKOK_ENGINE_H
```

```cpp
#include "Engine.h"
#include "thread"

void Engine::setLevel() {

    target = rand() % 6;
    target_text.setFont(font);
    target_text.setCharacterSize( size: 24);
    target_text.setString(std::to_string( val: target));
    target_text.setFillColor( color: Color::Black);
    target_text.setStyle(sf::Text::Bold);
    Vector2f position;
    position.x = 900;
    position.y = 300;
    target_text.setPosition(position);

    m_Circle.type = game_mode;
    target_num = 0;
    for(int i = 0; i < m_Circle.size[m_Circle.type]; i++)
    {
        bool tar= (rand() % 3)/2;
        target_num += tar;
        if(game_target_condition == no)
        {
            tar = 1;
        }
        m_Circle.round[i].update( target: tar);
        m_Circle.round[i].setPosition( type: game_mode, angle: 360.0/m_Circle.size[m_Circle.type] * i,  position: m_Circle.centerPosition,  &: m_Circle.segment);
        m_Circle.round[i].update( condition: rand() % 6);
        m_Circle.round[i].checkExecution( tar: target);
        m_Circle.round[i].setGameTargetCondition(game_target_condition);
        m_Circle.round[i].update( condition: m_Circle.round[i].getCondition());
    }
    target_num_text.setFont(font);
    target_num_text.setCharacterSize( size: 24);
    target_num_text.setString(std::to_string( val: target_num));
    target_num_text.setFillColor( color: Color::Black);
    target_num_text.setStyle(sf::Text::Bold);
    position.x = 960;
    position.y = 300;
    target_num_text.setPosition(position);
}
```

```cpp
Engine::Engine()
{
    Resolution = Vector2f( X: VideoMode::getDesktopMode().width,  Y: VideoMode::getDesktopMode().height);
    event_resolution = Vector2f( X: 0,  Y: 0);

    m_Window.create( mode: VideoMode( modeWidth: Resolution.x,  modeHeight: Resolution.y),  title: "Dtskok",  style: Style::Fullscreen);

    m_Window.setFramerateLimit( limit: 60);
    ImGui::SFML::Init( &: m_Window);

    m_BackgroundTexture.loadFromFile( filename: "resources/background.png");
    m_BackgroundSprite.setTexture( texture: m_BackgroundTexture);
    ball.loadFromFile( filename: "resources/ball.png");

    // загрузка шрифта для кириллицы
    ImGuiIO& io = ImGui::GetIO();
    io.Fonts->Clear();

    // нужно добавить файл со шрифтами и указать к нему путь
    io.Fonts->AddFontFromFileTTF( filename: "resources/arial.ttf",  size_pixels: 16.f,  font_cfg: NULL,
                                  glyph_ranges: ImGui::GetIO().Fonts->GetGlyphRangesCyrillic());
    font.loadFromFile( filename: "resources/arial.ttf");

    // фиксирование изменений
    ImGui::SFML::UpdateFontTexture();

    m_Ball.setSprite( &: ball);

    setLevel();

    //event_id = 3;
    //game_speed = 146;
}
```

```cpp
int Engine::menu()
{
    ImGui::Begin( name: "Меню");

    ShowFiles();

    const char* condition_game_name = (condition_game >= 0 && condition_game < 2) ? game_names[condition_game] : "Unknown";
    ImGui::SliderInt( label: "Состояние игры", v: &condition_game, v_min: 0, v_max: 1, format: condition_game_name);

    const char* condition_physicOverlay_name = (condition_physicOverlay >= 0 && condition_physicOverlay < 2) ? physicOverlay_names[condition_physicOverlay] : "Unknown";
    ImGui::SliderInt( label: "Физическая визуализация", v: &condition_physicOverlay, v_min: 0, v_max: 1, format: condition_physicOverlay_name);
    ImGui::DragInt( label: "Скорость игры (%)", v: &game_speed, v_speed: 1, v_min: 1, v_max: 200, format: "%d%%", flags: ImGuiSliderFlags_AlwaysClamp);

    const char* game_mode_name = (game_mode >= 0 && game_mode  < 2) ? game_mode_names[game_mode] : "Unknown";
    int old_game_mod = game_mode;
    ImGui::SliderInt( label: "Игровой режим", v: &game_mode, v_min: 0, v_max: 1, format: game_mode_name);
    const char* game_target_condition_name = (game_target_condition >= 0 && game_target_condition < 2) ? target_condition_names[game_target_condition] : "Unknown";
    int old_game_target_condition = game_target_condition;
    ImGui::SliderInt( label: "Конкретизация целевых секторов", v: &game_target_condition, v_min: 0, v_max: 1, format: game_target_condition_name);
    if(old_game_mod != game_mode || old_game_target_condition != game_target_condition)
    {
        setLevel();
    }

    if(ImGui::Button( label: "Выйти на рабочий стол"))
    {
        m_Window.close();
        return -1;
    }

    ImGui::End();

    return 0;
}

void Engine::start()
{
    while (m_Window.isOpen())
    {
        //studing();

        sf::Event sf_event;
        while (m_Window.pollEvent( &: sf_event))
        {
            ImGui::SFML::ProcessEvent( event: sf_event);
            if (sf_event.type == sf::Event::Closed)
                m_Window.close();
        }
```

```
            if (sf_event.type == sf::Event::Closed)
                m_Window.close();
        }

        Time dt = clock.restart();
        dt = dt*(game_speed*0.01f);
        float dtAsSeconds = dt.asSeconds();
        ImGui::SFML::Update( & m_Window, dt);
        if(condition_game == game_Open) {
            input();
            update(dtAsSeconds);
        }
        else{
            int result = menu();
            if(result == -1)
            {
                break;
            }
        }

        //ImGui::ShowDemoWindow();

        event(event_id);

        draw();
    }
    ImGui::SFML::Shutdown();
}

void Engine::studing()
{
    if(event_id == 3 || event_id == 4 || event_id == 5)
    {
        m_Ball.speed = Vector2f ( X: 0, Y: 0);
        event_id = 3;
        if(game_speed == 100)
        {
            event_id = 4;
            if(condition_physicOverlay == physicOverlay_Open)
            {
                event_id = 5;
                if(condition_game == game_Open)
                {
                    event_id = 0;
                    m_Ball.speed = Vector2f ( X: 100, Y: 0);
                }
            }
        }
    }
    else if(m_Ball.last_impact.impact_position != Vector2f( X: 0, Y: 0) && event_id != 6)
    {
        event_id = 6;
        condition_game = game_Close;
    }
}
```