Техническая документация к проекту snake-game

Автор: Yu-Leo

GitHub: snake-game

Краткое описание: README

Язык: C++ (стандарт C++17)

Рекомендуемая IDE: Visual Studio 2019 Используемые

библиотеки: SFML (v2.5.1)

Период разработки: июль – август 2021 г.

Описание игрового процесса (gameplay):

Классическая игра "Змейка".

После запуска игры открывается главное меню, в котором доступно 3 пункта: «Start new game» - начать новую игру, «Settings» - перейти в меню настроек, «Quit» - выйти из игры (закрыть окно). На заднем фоне отображается карта игрового поля без змейки и яблок. В правом верхнем углу окна – надпись с текущим счётом.

При выборе пункта «Start new game» игра запускается, появляется и начинает двигаться змейка, генерируется яблоко. Происходит непосредственно игровой процесс.

Игрок управляет длинным, тонким существом, напоминающим змею (отображается зелёными клетками), которое ползает по плоскости, собирая яблоки (красная или желтая клетки) и избегая столкновения с собственным хвостом и стенами. Каждый раз, когда змейка съедает яблоко, она становится длиннее, что постепенно усложняет игру. Игра заканчивается, когда голова змейки сталкивается с хвостом змейки либо врезается в стену. Стены отображаются голубыми клетками.

В игре присутствует 2 вида яблок: красные и желтые. Съедание красного яблока увеличивает длину змейки на одну клетку, а съедание желтого – на две.

Управление происходит при помощи *стрелок вверх, вниз, вправо, и влево* на клавиатуре. При нажатии *Escape* во время игры активируется пауза и

открывается меню паузы. При нажатии *Escape* в режиме паузы игра продолжается. Навигация по меню осуществляется при помощи *стрелок вверх и вниз*. Выбор пункта меню – *Enter*. Изменение значений в настройках – *стрелками влево и вправо*. Мышка в игре не задействована.

Меню, в режиме паузы содержит 3 пункта: «Resume game» - продолжить игру, «Settings и «Quit» - аналогичны соответствующим пунктам в главном меню. При изменении карты из меню в режиме паузы текущая игра прекращается и создаётся новая с выбранной картой.

При выборе пункта «Settings» открывается меню с настройками игры, в котором можно изменить уровень громкости внутриигровых звуков, поменять режим скорости, или выбрать карты игрового поля.

На выбор доступно 5 различных карт игрового поля.

Присутствует 2 различных режима скоростей: автоматический – увеличение скорости в зависимости от текущего счёта и ручной – задание скорости в условных единицах от 1 до 6, где 1 – медленно, 6 – быстро.

Описание настроек:

«Volume: 10» - уровень громкости внутриигровых звуков. Изменяется от 0% до 100% с шагом 5%. «Speed: Auto» - режим скорости: автоматический (Auto) либо ручной (значение от 1 до 6).

«Map: 0» - номер карты игрового поля (от 0 до 4).

Описание механики игры:

Игровое поле хранится в виде матрицы, на которой пустые клетки отмечаются *нулем*; клетки, в которых находится змейка – от *длина змейки* до *1*. На каждой итерации происходит изменение направления движения змейки при необходимости, после чего осуществляется движение змейки за счёт последовательного перемещения позиции головы на соседнюю клетку, уменьшения значений во всех клетках, принадлежащих змейке, на 1, и инициализацией клетки, в которой находится голова, значением, равным длине змейки.

Обработка команд по изменению направления движения змейки происходит следующим образом: при нажатии клавиш со стрелками на клавиатуре, если

желаемое изменение направления возможно, команда по изменению направления добавляется в очередь команд, из которой на каждой итерации обрабатываются команды (в порядке очереди, FIFO).
Размеры поля неизменны и равны 35 * 20 клеток (*ширина* * *высота*).

Возможно добавление новых карт путем добавления файлов конфигураций карт в папку **maps**. При этом необходимо соблюдать нумерацию: имена файлов должны иметь вид «map*N*.txt», где *N* – порядковый номер карты при нумерации с 0. После чего необходимо изменить общее кол-во карт в файле **MapsList.h** в поле `NUMBER_OF_MAPS`.

В случае, если приложение аварийно завершается с ошибкой, возможно, проблема в том, что файл с настройками (settings.txt) был изменён вручную, его необходимо удалить.

Список модулей:

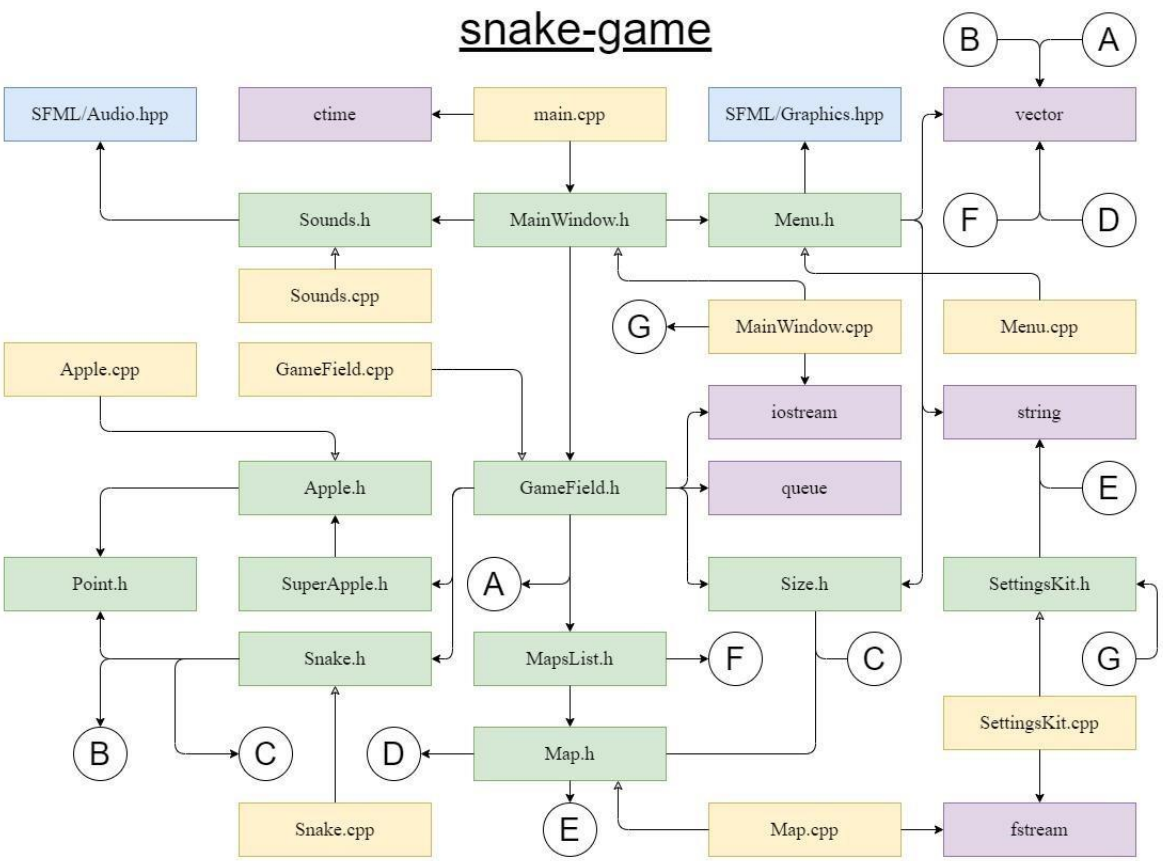
Механика игры:

- **GameField** (GameField.h, GameField.cpp) – игровое поле.
- **Snake** (Snake.h, Snake.cpp) – игрок (змейка).
- **Apple** (Apple.h, Apple.cpp) – яблоко, располагающееся на игровом поле.
- **SuperApple** (SuperApple.h) – супер-яблоко, располагающееся на игровом поле.
- **Point** (Point.h) – точка игрового поля.
- **MapsList** (MapsList.h) – список возможных карт игрового поля.
- **Map** (Map.h, Map.cpp) – карта игрового поля.
- **Size** (Size.h) – размеры игрового поля.
- **SettingsKit** (SettingsKit.h, SettingsKit.cpp) – набор игровых параметров.

Графический интерфейс и звуки:

- **Main** (main.cpp) – главный запускаемый файл проекта.
- **MainWindow** (MainWindow.h, MainWindow.cpp) – главное окно игры.
- **Menu** (Menu.h, Menu.cpp) – меню, используемое в игре.
- **Sounds** (Sounds.h, Sounds.cpp) – звуки, используемые в игре.

Диаграмма зависимостей: (.drawio, .pdf)



Описание модулей:

Механика игры: GameField:

Сущность игрового поля. Хранит в себе матрицу, на которой отмечены клетки, в которых находится змейка, яблоко и стены. Обрабатывает входящие команды, движение и изменение длины змейки, генерацию яблок, режимы старта, паузы, игры, и финиша.

Snake:

Сущность змейки. Хранит в себе позицию головы змейки, длину и направление. Выполняет движение головы, смену направления и удлинение.

Apple:

Сущность яблока. Хранит в себе координаты.

SuperApple:

Сущность супер-яблока. Полностью аналогична обычному яблоку.

Point:

Структура клетки игрового поля. Содержит координаты,

поддерживает операцию сравнения на равенство. MapsList:

Сущность списка карт игрового поля. Читает карты из файлов и хранит их в виде массива объектов класса **Map**. Поддерживает операцию взятия по индексу.

Map:

Сущность карты игрового поля. Хранит размер карты и двумерный массив, в котором отмечены пустые клетки и клетки, в которых находятся стены. В том числе отвечает (статический метод) за чтение карты из файла и преобразование в объект класса **Map**.

Size:

Структура размеров прямоугольной сущности. Содержит ширину и высоту, поддерживает операции сравнения на равенство и неравенство.

SettingsKit:

Структура-набор настроек игры: уровень громкости, скорость, выбранная карта игрового поля. Выполняет операции сохранения настроек в файл и чтения их из файла.

Графический интерфейс и звуки:

Main:

Управление игровым процессом. В цикле происходит ожидание ввода команды для управления змейкой (стрелки на клавиатуре). После чего происходит обработка всех событий, перерисовка игрового поля и его отображение в окне игры.

MainWindow:

Сущность окна игры. Отвечает за отрисовку самого окна, всех игровых элементов и обработку нажатий на клавиатуру.

Menu:

Сущность внутриигрового меню. Отвечает за отображение меню.

Sounds:

Модуль с внутриигровыми звуками. Выполняет операции воспроизведения звука, позволяет регулировать громкость звуков.

Список классов и описание их полей и методов:

Механика игры: GameField:

#define CHANCE_OF_GENERATING_SUPER_APPLE 15 – шанс генерации супер яблока в процентах. **static Size DEFAULT_SIZE = Size(35, 20);** - размер поля по умолчанию. **const MapsList MAPS(DEFAULT_SIZE);** - список карт игрового поля.

class GameField { public:

enum class GameStatus { - статусы игры.

STARTED, - игра запущена.

ACTIVE, - игровой процесс выполняется.

PAUSE, - игра на паузе.

FINISHED – игра завершена.

};

enum class CellTypes { - типы ячеек.

NONE,

APPLE,

SUPER_APPLE,

SNAKE_HEAD,

SNAKE_BODY,

WALL

};

enum class Collisions { - типы коллизий головы змейки.

APPLE, - с обычным яблоком.

SUPER_APPLE, - с супер яблоком.

BODY, - с телом.

WALL, - со стенами

NONE – отсутствие коллизий.

};

GameField(const Size& size, int map_number=0, bool only_walls=false); - конструктор поля с заданными размерами, картой и генерацией либо только стен, либо стен, змейки, и ябока. Инициализация поля, генерация яблока, отрисовка яблока и змейки (render_apple, render_apple).

GameField(); - конструктор по умолчанию с дефолтными размерами.

void one_iteration(); - обработка одной игровой итерации.

void key_pressed(); - обновление последней введённой команды.

Необходимо вызывать перед обработкой и добавлением новых команд управления в очередь команд.

void insert_command(int direction); - добавление новой команды по смене направления движения змейки в очередь команд. **void start();** - начать игру. **void finish();** - завершить игру. **void pause();** - поставить на паузу.

void unpause(); - снять с паузы.

void clear_collision(); - очистка поля коллизий.

Collisions get_collision() const; - геттер для получения коллизий.
int get_score() const; - геттер для счёта.

Size get_size() const; - геттер размеров игрового поля.

int get_cells_without_walls() const; - получить кол-во клеток на игровом поле без стен. **int get_snake_direction() const;** - геттер для направления движения змейки.

GameStatus get_game_status() const; - геттер статуса игры

CellTypes get_cell_type(const Point& point) const; - геттер типа ячейки.

private:

static const int FIELD_CELL_TYPE_NONE = 0; - значения в пустых ячейках.

static const int FIELD_CELL_TYPE_APPLE = -1; - значение в ячейке с яблоком.

static const int FIELD_CELL_TYPE_SUPER_APPLE = -2; - значение в ячейке с супер-яблоком.

static const int FIELD_CELL_TYPE_WALL = -3; - значения в ячейках со стенами.

int score = 0; - счёт.

Size size; - размеры игрового поля.

std::vector<std::vector<char>> field; - матрица игрового поля.

Snake snake; - сама змейка (объект класса Snake).

Apple apple; - яблоко.

std::queue<int> snake_directions; - очередь команд управления змейкой.

int last_snake_direction; - предыдущее направление змейки.

GameStatus game_status; - статус игры.

Collisions collision = Collisions::NONE; - коллизии головы змейки.

void init_field(); - инициализация игрового поля (заполнение значениями пустой ячейки).

void resize_matrix(); - установка размеров матрицы игрового поля.

void set_map(const Map& map); - установка стен на карту.

void render_snake(); - отрисовка змейки на игровом поле (заполнение клеток, в которых находится змейка символами для обозначения тела змейки).

int count_cells_without_walls(); - счёт кол-ва клеток на игровом поле без стен.

void move_snake(); - движение змейки: изменение координаты головы, проверка столкновений, изменение значений в клетках тела змейки.

void turn_snake(int direction); - смена направления движения змейки (делегирование объекту класса Snake).

void check_collisions(); - проверка на столкновение головы змейки с другими клетками. **void increase_length();** - увеличение змейки и счёта. **void generate_new_apple();** - генерация нового яблока. **void grow_snake();** - увеличение значений в клетках змейки. **void decrease_snake_cells();** - уменьшение значений в клетках змейки. **bool is_cell_empty(const Point& cell);** - проверка клетки на пустоту. **int count_empty_cells();** - получение количества пустых клеток. **Point get_random_empty_cell();** - получение случайной пустой ячейки.

};

Snake: **class Snake { public: enum Directions { -** константы направлений движения змейки.

RIGHT,

DOWN,

LEFT,

UP };

Snake(); - конструктор по умолчанию (пустой).

void move_head(); - изменение координаты головы в зависимости от заданного направления. **void set_field_size(Size field_size);** - инициализация размеров поля.

void increase_length(); - увеличение длины на одну ячейку.

void change_direction(int new_direction); - смена направления.

Point get_head_pos() const; - геттер позиции головы.

int get_length() const; - геттер длины. **int**

get_direction() const; - геттер направления.

private: int length = 3;

- длина.

Point head_position = Point(this->length – 1 + 3, 2); - позиция головы.

int direction = Directions::RIGHT; - направление движения. **Size**

field_size; - размеры поля, на котором используется объект класса.

Необходимо инициализировать фактическим значением!

};

Apple: **class**

Apple {

public:

Apple(); - конструктор по умолчанию. Использует конструктор по умолчанию структуры Point.

Apple(const Point& point); - конструктор с указанием конкретной точки, в которой необходимо создать яблоко.

Point get_coordinates(); - геттер для поля coordinates.

protected:

Point coordinates; - точка, в которой находится яблоко.

};

SuperApple:

class SuperApple : public Apple { public:

SuperApple() : Apple() {}

SuperApple(const Point& point) : Apple(point) {}

};

Point:

struct Point { **int x = 0;** - x-координата точки. **int y = 0;** - у-координата точки.

Point(int x, int y); - конструктор с произвольным заданием координат

Point() : Point(0, 0) {}; - конструктор по умолчанию.

bool operator==(const Point& other); - перегрузка оператора равенства. Точки равны, если равны их координаты.

};

MapsList:

class MapsList { public:

const int NUMBER_OF_MAPS = 5; - общее количество карт игрового поля.

MapsList(const Size &size); - конструктор с чтением карт из файлов.

Map operator[](int index) const; - перегрузка оператора [] для получения карты по её индексу в списке. **private:** **std::vector<Map> maps;** - массив карт игрового поля.

};

Map:

struct Map { static const int WALL = 1; - значение для стены.

static const int NONE = 0; - значение для пустой клетки.

std::vector<std::vector<int>> map; - сама карта.

Size size; - размеры карты. **static Map read_from_file(const std::string& file_name);** - функция чтения карты из файла.

Map(const Size& size = Size(), const std::vector<std::vector<int>>& map = {}); - конструктор по размерам и матрице с обозначениями стен.

};

Size:

struct Size { int width; - ширина игрового поля. **int height;** - высота игрового поля.

Size(int w, int h); - конструктор с произвольным заданием координат.

Size() : Size(0, 0) {}; - конструктор по умолчанию. **bool operator==(const Size& other) const;** - перегрузка оператора равенства.

bool operator!=(const Size& other) const; - перегрузка оператора не равенства.

};

SettingsKit:

const std::string FILE_NAME = "settings.txt"; - имя файла с настройками.

struct SettingsKit { int volume; - уровень громкости.

std::string speed_item; - режим скорости.

int map_number; - номер карты.

SettingsKit();

SettingsKit(int volume, std::string speed_item, int map_number);

void save_to_file(); - сохранение набора настроек в файл настроек.

static SettingsKit load_from_file(); - чтение набора настроек из файла.

};

Графический интерфейс и звуки: MainWindow:

#define CELL_SIZE 32 – размер одной ячейки в пикселях.

#define TOP_PADDING 55 – верхний отступ от края окна до игрового поля.

class MainWindow : public sf::RenderWindow { - класс главного окна игры.

public:

MainWindow(const Size& size); - конструктор с устанавливаемыми размерами. **void event_handling();** - обработчик событий.

void one_iteration(); - выполнение одной игровой итерации.

void redraw(); - перерисовка игрового окна.

void delay(); - задержка после выполнения одной итерации.

private:

class Speed { - класс для контроля скорости игры.

public:

void delay(); - выполнение задержки в зависимости от скорости.

int get_num() const; - геттер для номера скорости.

std::string get_active_item() const; - геттер активного пункта настроек.

void update(const MainWindow& window); - обновление скорости (при автоматическом режиме).

void set_speed(std::string speed_item); - принудительная установка активного пункта в настройках. **void increase_speed();** - переход к следующему пункту в настройках. **void reduce_speed();** - переход к предыдущему пункту в настройках. **private:**

std::vector<std::string> speed_items = { "Auto", "1", "2", "3", "4", "5", "6" }; - пункты в настройках. **int active_speed_item = 0;** - индекс активного пункта.

std::vector<int> delays = { 130, 110, 100, 90, 80, 65 }; - временные задержки (в миллисекундах).

bool auto_speed = true; - включен ли режим автоматического изменения скорости. **int speed = 0;** - номер скорости (0 – медленно, 5 – быстро).

bool is_correct_speed_item(std::string speed_item); - проверка пункта настроек на корректность.

};

Speed speed; - скорость игры.

int map_number = 0; - номер выбранной карты игрового поля.

Size window_size; - размер окна в пикселях.

GameField game_field; - механика игры (игровое поле)

Size game_field_size; - размеры игрового поля в клетках

struct Textures { - используемые текстуры.

sf::Texture none, apple, super_apple, snake_head, snake_body, wall;

};

struct Sprites { - используемые спрайты.

sf::Sprite none, apple, super_apple, snake_head, snake_body, wall;

};

Textures textures;

Sprites sprites;

Sounds sounds; - звуки игры.

sf::Font font; - шрифт для надписей.

sf::Text score_text; - надпись с счётом.

sf::Text game_over_text; - используемые текстуры.

struct MenuList { - список меню.

public: enum ActiveMenu { - название активного меню.

NONE,

MAIN,

PAUSE,

SETTINGS

};

int active = MAIN; - активное меню.

Menu main;

Menu pause;

Menu settings;

MenuList();

void draw(MainWindow& window); - отрисовка активного меню в окне игры.

void operations(MainWindow& window); - операции, выполняемые из меню.

void next_item(); - переключение на следующий элемент в меню.

void previous_item(); - переключение на предыдущий элемент в меню.

private:

Операции, выполняемые из различных меню.

void main_menu_operations(MainWindow& window);

void pause_menu_operations(MainWindow& window);

void settings_menu_operations(MainWindow& window);

};

MenuList menu; - список меню.

const sf::Color BACKGROUND_COLOR = sf::Color(0, 0, 0); - цвет фона игры.

void load_textures(); - загрузка текстур из файлов. **void set_textures();** - установка текстур на спрайты. **void set_text_settings();** - установка шрифтовых настроек на надписи. **void set_score_text_color();** - установка цвета надписи со счётом. **void**

handling_control(const sf::Event& event); - обработка команд управления змейкой.

void handling_menu_navigation(const sf::Event& event); - обработка команд навигации по меню.

void change_map(int new_map_num); - смена карты игрового поля.

void play_sounds(); - воспроизведение звуков коллизий.

void draw_screen(); - отрисовка игрового поля и надписи со счётом.

void draw_field(); - отрисовка игрового поля. **void draw_cell(const Point& point);** - отрисовка одной ячейки. **void rotate_snake_head_sprite();** - поворот спрайта головы змеи. **void draw_score_bar();** - отрисовка надписи со счётом.

};

Menu: **class Menu {** - класс меню, используемого в игре.

public: Menu(); void set_text_to_items(const std::vector<std::string>& items); - установка надписей на пунктах меню.

void set_text_to_item(int index, const std::string& text); - установка надписи на конкретном пункте меню.

void draw(sf::RenderWindow &window); - отображение меню в окне.

void next_item(); - переход к следующему пункту меню.

void previous_item(); - переход к предыдущему пункту меню.

void reset_active_item(); - сброс активного эл-та на нулевой (первый пункт).

int get_active_item_index(); - геттер индекса активного пункта меню.

private: struct Position { - позиция элемента меню. **float x = 0; float y = 0; Position() {}**

Position(float x, float y) {

this->x = x; this->y = y;

}

};

const float HORIZONTAL_PADDING = 40.0; - горизонтальный отступ.

const float VERTICAL_PADDING = 30.0; - вертикальный отступ. **const float ITEM_PADDING = 20;** - отступ между пунктами меню. **sf::Font font;** - шрифт, используемый в меню.

const sf::Color BACKGROUND_COLOR = sf::Color(220, 220, 220); - цвет фона.

const sf::Color INACTIVE_TEXT_COLOR = sf::Color(128, 128, 128); - цвет неактивных пунктов.

const sf::Color ACTIVE_TEXT_COLOR = sf::Color::Black; - цвет активного пункта.

std::vector<sf::Text> menu_items; - пункты меню.

std::vector<std::string> menu_items_text; - надписи на пунктах меню.

int active_item_index = 0; - индекс активного пункта меню.

void load_font(); - загрузка шрифта из файла.

void set_font_settings(); - установка шрифтовых настроек на элементы меню. **void set_titles();** - установка надписей на элементы меню.

void draw_background(sf::RenderWindow& window, const Size& size, const Position& pos); - отображение заднего фона меню.

void draw_items(sf::RenderWindow& window, const Position& bg_pos); - отображение пунктов меню.

Size get_background_size(); - расчёт размеров заднего фона меню.

};

Sounds:

class Sounds { public: enum

SoundNames { - названия звуков.

ATE_APPLE,

COLLISION_WITH_BODY,

COLLISION_WITH_WALL,

MENU_NAVIGATE

};

Sounds(); void play(int sound_name); - воспроизведение звука по его названию. **void set_volume(int volume);** - сеттер для громкости.

int get_volume(); - геттер для громкости.

void turn_up_volume(); - увеличение громкости.

void turn_down_volume(); - уменьшение громкости.

private:

const int MAX_VOLUME = 100; - максимальная громкость.

const int MIN_VOLUME = 0; - минимальная громкость.

int volume = 10; - текущее значение громкости.

struct SoundBuffers {

sf::SoundBuffer ate_apple;

sf::SoundBuffer collision_with_wall;

sf::SoundBuffer collision_with_body;

sf::SoundBuffer menu_navigate;

};

SoundBuffers sound_buffers;

sf::Sound ate_apple; sf::Sound

collision_with_wall; sf::Sound

collision_with_body; sf::Sound

menu_navigate;

void load_sound_buffers(); - загрузка звуков из файлов.

void set_sound_buffers(); - установка звуков.

void set_volume_to_sounds(); - установка громкости на объекты звуков. Необходимо вызывать после изменения поля this->volume.

};