

# 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 врежется 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:

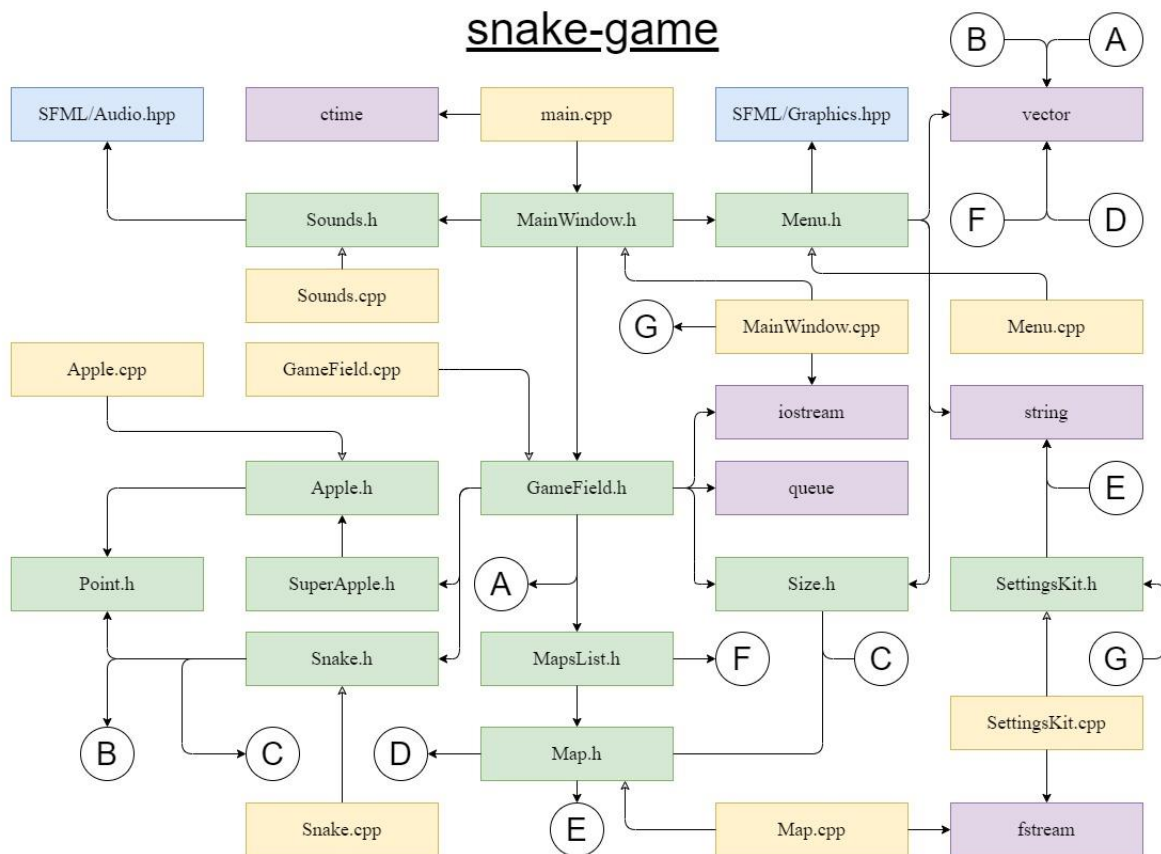
#### *Механика игры:*

- **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. Обработывает 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 яблока. 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 введенной 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 unpauses();** - 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 (делегирующее 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; - y-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`.

```
};
```