Техническая документация к проекту snake-game

Автор: Yu-Leo

GitHub: snake-game

Язык: C++ (стандарт C++17).

Используется библиотека SFML.

Модули:

- **Main** (main.cpp) – главный запускаемый файл проекта.
- **Point** (Point.h) – точка игрового поля.
- **Size** (Size.h) – размеры игрового поля.
- **Apple** (Apple.h, Apple.cpp) – яблоко, располагающееся на игровом поле.
- **Snake** (Snake.h, Snake.cpp) – игрок (змейка).
- **GameField** (GameField.h, GameField.cpp) – игровое поле.
- **Menu** (Menu.h, Menu.cpp) – меню, используемое в игре.
- **Sounds** (Sounds.h, Sounds.cpp) – звуки, используемые в игре
- **MainWindow** (MainWindow.h, MainWindow.cpp) – главное окно игры.

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.

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 сравнения. Точки равны, если равны их координаты.

};

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.

};

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.

private:

Point coordinates; - точка, в которой находится яблоко.

};

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.

Point get_head_pos() const; - геттер позиции головы.

int get_length() const; - геттер длины.

int get_direction() const; - геттер направления.

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

void change_direction(int new_direction); - смена направления.

private:

int length = 3; - длина.

Point head_position = Point(this->length - 1, 0); - позиция головы.

int direction = Directions::RIGHT; - направление.

Size field_size; - размеры поля, на котором используется объект класса. Необходимо инициализировать фактическим значением!

};

GameField:

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,

SNAKE,

WALL

};

enum class Collisions { - типы коллизий головы змейки.

APPLE, - с яблоком.

BODY, - с телом.

WALL, - со стенами

NONE – отсутствие коллизий.

};

GameField(const Size& size); - конструктор поля с заданными размерами. 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.

int get_score() const; - геттер для счёта.

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

GameStatus get_game_status() const; - геттер статуса игры

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

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

void clear_collision(); - очистка поля коллизий.

private:

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

static const int FIELD_CELL_TYPE_APPLE = -1; - значение в ячейках с яблоками.

static const int FIELD_CELL_TYPE_WALL = -2; - значения в ячейках со стенами.

enum class Symbols { - символы для отображения игрового поля в консоли (терминале).

SNAKE = '#',

NONE = '-',

APPLE = '*',

WALL = '$'

};

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 resize_matrix(); - установка размеров матрицы игрового поля.

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

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 grow_snake(); - увеличение значений в клетках змейки.

void decrease_snake_cells(); - уменьшение значений в клетках змейки.

void set_walls(); - установка стен на игровом поле.

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

void render_apple(); - отрисовка яблока на игровом поле (заполнение клетки, в которой находится яблоко символом для обозначения яблока).

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.

friend void print_cell(std::ostream& out, const GameField& game_field, const Point& cell); - дружественная функция для вывода одной клетки игрового поля в консоль (терминал).

friend std::ostream& operator<< (std::ostream& out, const GameField& game_field); - дружественная функция для вывода матрицы игрового поля в консоль (терминал).

};

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.

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_texts(); - установка надписей на элементы меню.

Size get_background_size(); - расчёт размеров заднего фона 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.

};

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.
   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(); - установка громкости звуков.

};

MainWindow:

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:

int speed = 2; - скорость игры. 0 –медленная, 4 – быстрая

bool field_regeneration = false; - должно ли быть перерисовано поле при старте

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, snake, wall;

};

struct Sprites { - используемые спрайты.

sf::Sprite none, apple, snake, 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 play_sounds(); - воспроизведение звуков коллизий.

void draw_cell(const Point& point); - отрисовка одной ячейки.

void draw_field(); - отрисовка игрового поля.

void draw_score_bar(); - отрисовка надписи со счётом.

void draw_screen(); - отрисовка игрового поля и 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.

};