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.
- **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.

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.

ON, - активна.

OFF – завершена.

};

enum class CellTypes { - типы ячеек.

NONE,

APPLE,

SNAKE

};

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 finish_game(); - завершить игру.

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.

private:

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

SNAKE = '#',

NONE = '-',

APPLE = '*'

};

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.

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.

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 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).

};

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.

private:

GameField game_field; - игровое поле.

void draw_cell(const Point& point, sf::RectangleShape& cell); - отрисовка одной ячейки на поле.

void draw_field(); - отрисовка игрового поля в окне.

};