Техническая документация к проекту snake-game

Автор: Yu-Leo

GitHub: snake-game

Язык: C++ (стандарт C++17)

Модули:

- **Main** (main.cpp) – главный запускаемый файл проекта.
- **Constants** (constants.h) – константы проекта.
- **Point** (Point.h, Point.cpp) – точка игрового поля.
- **RandomPoint** (RandomPoint.h, RandomPoint.cpp) – случайная точка 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.

Main:

Управление игровым процессом. В цикле происходит ожидание ввода команды для управления змейкой (wasd),в случае ввода любой другой команды происходит окончание игры. Для ввода команды необходимо нажать Enter. После чего происходит обработка всех событий, перерисовка игрового поля и его вывод в консоль (терминал).

Constants:

const int GAME_FIELD_SIZE = 20; - размеры игрового поля.

namespace Directions { - константы для направлений движения змейки.

```
    const int RIGHT = 0;
    const int DOWN = 1;
    const int LEFT = 2;
    const int UP = 3;
}
```

Point:

struct Point {

int x = 0; - x-координата точки.

int y = 0; - y-координата точки.

Point(); - конструктор по умолчанию (x = 0; y = 0).

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

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

};

RandomPoint:

struct RandomPoint : public Point {

RandomPoint(); - конструктор по умолчанию. Координаты точки - случайные числа в пределах размеров игрового поля.

RandomPoint(const std::vector<Point>& acceptable_points); - конструктор с указанием недопустимых точек. Координаты сточки – случайные числа в пределах размеров игрового поля за исключением пар, образующих недопустимые точки

};

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:

Snake(); - конструктор по умолчанию. Инициализирует змейку (init_snake_dots) с размером и направлением по умолчанию (DEFAULT_SIZE, Directions::RIGHT).

Point get_point_by_index(int index); - геттер для получения координат точки змейки по её индексу в массиве с координатами точек змейки.

int get_size(); - геттер для размера змейки.

void move(); - сдвиг змейки на одну клетку в установленном направлении.

void change_direction(int new_direction); - смена направления движения змейки (используются константы Direction::)

bool check_collision_with_body(); - проверка на совпадение координат головы змейки и какой либо клетки её тела.

void increase_size(); - увеличение длины змейки на одну клетку.

std::vector<Point> get_points(); - геттер для координат точек змейки.

private:

static const int DEFAULT_SIZE = 3; - размер змейки по умолчанию.

Point dots[GAME_FIELD_SIZE * GAME_FIELD_SIZE]; - массив с координатами точек тела змейки.

int size; - размер змейки.

int direction; - текущее направление (используются константы Direction::).

void init_snake_dots(); - инициализация змейки в начальных координатах (в один ряд, горизонтально, последняя точка находится в координатах (0; 0).

bool is_change_of_direction_correct(int new_direction); - проверка корректности смены направления (используется в change_direction).

};

GameField:

class GameField {

public:

GameField(); - конструктор по умолчанию. Инициализация поля (init_field), генерация яблока, отрисовка яблока и змейки (render_apple, render_apple).

void update(); - перерисовка яблока и змейки на поле.

void move_snake(); - движение змейки (делегирование объекту класса Snake).

void check_collision_with_apple(); - проверка на совпадение координат головы змейки и яблока. Если совпадение есть, длина змейки увеличивается и генерируется новое яблоко.

bool is_game_over(); - проверка на проигрыш (столкновение змейки с самой собой или границами игрового поля).

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

private:

static const char SNAKE_SYMBOL = '#'; - символ для обозначения тела змейки на игровом поле.

static const char VOID_SYMBOL = '-'; - символ для обозначения пустой клетки на игровом поле.

static const char APPLE_SYMBOL = '*'; - символ для обозначения яблока на игровом поле.

char field[GAME_FIELD_SIZE][GAME_FIELD_SIZE]; - матрица игрового поля.

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

Apple apple; - яблоко.

void init_field(); - инициализация игрового поля (заполнение символами пустой клетки).

bool check_collision_with_snake_body(); - проверка на столкновение змейки с самой собой (делегирование объекту класса Snake).

bool check_collision_with_borders(); - проверка на столкновение змейки с границами игрового поля.

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).

std::vector<Point> generate_acceptable_points_for_new_apple(); - генерация списка клеток, в которых можно сгенерировать новое яблоко (все клетки игрового поля за исключением клеток, в которых находится змейка).

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

};