#include <SFML/Graphics.hpp>

#include <cstdlib>

#include <vector>

#include <iostream>

using namespace std;

constexpr auto FIELD\_CELL\_TYPE\_NONE = 0;

constexpr auto FIELD\_CELL\_TYPE\_APPLE = -1;

constexpr auto FIELD\_CELL\_TYPE\_WALL = -2;

constexpr auto SNAKE\_DIRECTION\_UP = 0;

constexpr auto SNAKE\_DIRECTION\_RIGHT = 1;

constexpr auto SNAKE\_DIRECTION\_DOWN = 2;

constexpr auto SNAKE\_DIRECTION\_LEFT = 3;

const int field\_size\_x = 35;

const int field\_size\_y = 25;

const int cell\_size = 32;

const int window\_width = field\_size\_x \* cell\_size;

const int window\_height = field\_size\_y \* cell\_size;

int field[field\_size\_y][field\_size\_x];

int snake\_position\_x = field\_size\_x / 2;

int snake\_position\_y = field\_size\_y / 2;

int snake\_length = 4;

int snake\_direction = SNAKE\_DIRECTION\_RIGHT;

int score = 0;

bool game\_over = false;

bool game\_paused = false;

int difficulty\_level()

{

int snake\_speed, level;

cout << "Вашему вниманию представлена игра SNAKE" << endl << endl;

cout << "Для управления используйте курсоры, для быстрого выхода - нажмите escape" << endl;

cout << "Чтобы поставить игру на паузу - нажмите пробел, для продолжения используйте курсоры или enter" << endl;

cout << endl << "Доступны уровни сложности: " << endl;

cout << "0 - standart" << endl;

cout << "1 - easy" << endl;

cout << "2 - medium" << endl;

cout << "3 - hard" << endl;

cout << "4 - crazy" << endl;

cout << "5 - impossible" << endl;

cout << "6 - RF" << endl;

cout << endl << "Выберите уровеень сложности: "; cin >> level; cout << endl;

switch (level) {

case 1:

snake\_speed = 100;

break;

case 2:

snake\_speed = 80;

break;

case 3:

snake\_speed = 60;

break;

case 4:

snake\_speed = 40;

break;

case 5:

snake\_speed = 20;

break;

case 6:

snake\_speed = 0;

break;

default:

snake\_speed = 120;

}

return snake\_speed;

}

int get\_random\_empty\_cell()

{

int empty\_cell\_count = 0;

for (int j = 0; j < field\_size\_y; j++) {

for (int i = 0; i < field\_size\_x; i++) {

if (field[j][i] == FIELD\_CELL\_TYPE\_NONE) {

empty\_cell\_count++;

}

}

}

int target\_empty\_cell\_index = rand() % empty\_cell\_count;

int empty\_cell\_index = 0;

for (int j = 0; j < field\_size\_y; j++) {

for (int i = 0; i < field\_size\_x; i++) {

if (field[j][i] == FIELD\_CELL\_TYPE\_NONE) {

if (empty\_cell\_index == target\_empty\_cell\_index) {

return j \* field\_size\_x + i;

}

empty\_cell\_index++;

}

}

}

return -1;

}

void add\_apple()

{

int apple\_pos = get\_random\_empty\_cell();

if (apple\_pos != -1) {

field[apple\_pos / field\_size\_x][apple\_pos % field\_size\_x] = FIELD\_CELL\_TYPE\_APPLE;

}

}

void clear\_field()

{

for (int j = 0; j < field\_size\_y; j++) {

for (int i = 0; i < field\_size\_x; i++) {

field[j][i] = FIELD\_CELL\_TYPE\_NONE;

}

}

for (int i = 0; i < snake\_length; i++)

field[snake\_position\_y][snake\_position\_x - i] = snake\_length - i;

for (int i = 0; i < field\_size\_x; i++) {

if (i < 10 || field\_size\_x - i - 1 < 10) {

field[0][i] = FIELD\_CELL\_TYPE\_WALL;

field[field\_size\_y - 1][i] = FIELD\_CELL\_TYPE\_WALL;

}

}

for (int j = 1; j < field\_size\_y - 1; j++) {

if (j < 8 || field\_size\_y - j - 1 < 8) {

field[j][0] = FIELD\_CELL\_TYPE\_WALL;

field[j][field\_size\_x - 1] = FIELD\_CELL\_TYPE\_WALL;

}

}

add\_apple();

}

void draw\_field(sf::RenderWindow &window)

{

sf::Texture none\_texture;

none\_texture.loadFromFile("images/none.png");

sf::Sprite none;

none.setTexture(none\_texture);

sf::Texture snake\_texture;

snake\_texture.loadFromFile("images/snake.png");

sf::Sprite snake;

snake.setTexture(snake\_texture);

sf::Texture apple\_texture;

apple\_texture.loadFromFile("images/apple.png");

sf::Sprite apple;

apple.setTexture(apple\_texture);

sf::Texture wall\_texture;

wall\_texture.loadFromFile("images/wall.png");

sf::Sprite wall;

wall.setTexture(wall\_texture);

for (int j = 0; j < field\_size\_y; j++) {

for (int i = 0; i < field\_size\_x; i++) {

switch (field[j][i]) {

case FIELD\_CELL\_TYPE\_NONE:

none.setPosition(float(i \* cell\_size), float(j \* cell\_size));

window.draw(none);

break;

case FIELD\_CELL\_TYPE\_APPLE:

apple.setPosition(float(i \* cell\_size), float(j \* cell\_size));

window.draw(apple);

break;

case FIELD\_CELL\_TYPE\_WALL:

wall.setPosition(float(i \* cell\_size), float(j \* cell\_size));

window.draw(wall);

break;

default:

snake.setPosition(float(i \* cell\_size), float(j \* cell\_size));

window.draw(snake);

}

}

}

}

void grow\_snake()

{

for (int j = 0; j < field\_size\_y; j++) {

for (int i = 0; i < field\_size\_x; i++) {

if (field[j][i] > FIELD\_CELL\_TYPE\_NONE) {

field[j][i]++;

}

}

}

}

void make\_move()

{

switch (snake\_direction) {

case SNAKE\_DIRECTION\_UP:

snake\_position\_y--;

if (snake\_position\_y < 0) {

snake\_position\_y = field\_size\_y - 1;

}

break;

case SNAKE\_DIRECTION\_RIGHT:

snake\_position\_x++;

if (snake\_position\_x > field\_size\_x - 1) {

snake\_position\_x = 0;

}

break;

case SNAKE\_DIRECTION\_DOWN:

snake\_position\_y++;

if (snake\_position\_y > field\_size\_y - 1) {

snake\_position\_y = 0;

}

break;

case SNAKE\_DIRECTION\_LEFT:

snake\_position\_x--;

if (snake\_position\_x < 0) {

snake\_position\_x = field\_size\_x - 1;

}

break;

}

if (field[snake\_position\_y][snake\_position\_x] != FIELD\_CELL\_TYPE\_NONE) {

switch (field[snake\_position\_y][snake\_position\_x]) {

case FIELD\_CELL\_TYPE\_APPLE:

score++;

snake\_length++;

grow\_snake();

add\_apple();

break;

case FIELD\_CELL\_TYPE\_WALL:

game\_over = true;

break;

default:

if (field[snake\_position\_y][snake\_position\_x] > 1) {

game\_over = true;

}

}

}

for (int j = 0; j < field\_size\_y; j++) {

for (int i = 0; i < field\_size\_x; i++) {

if (field[j][i] > FIELD\_CELL\_TYPE\_NONE) {

field[j][i]--;

}

}

}

field[snake\_position\_y][snake\_position\_x] = snake\_length;

}

int main(void)

{

int speed, level;

setlocale(0, "");

system("color B0");

srand(time(NULL));

speed = difficulty\_level();

sf::RenderWindow window(sf::VideoMode(window\_width, window\_height), "snake", sf::Style::Close);

clear\_field();

vector<int> snake\_direction\_queue;

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

if (event.type == sf::Event::KeyPressed) {

int snke\_direction\_last = snake\_direction\_queue.empty() ? snake\_direction : snake\_direction\_queue.at(0);

switch (event.key.code) {

case sf::Keyboard::Up:

game\_paused = false;

if (snke\_direction\_last != SNAKE\_DIRECTION\_DOWN) {

if (snake\_direction\_queue.size() < 2) {

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), SNAKE\_DIRECTION\_UP);

}

}

break;

case sf::Keyboard::Right:

game\_paused = false;

if (snke\_direction\_last != SNAKE\_DIRECTION\_LEFT) {

if (snake\_direction\_queue.size() < 2) {

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), SNAKE\_DIRECTION\_RIGHT);

}

}

break;

case sf::Keyboard::Down:

game\_paused = false;

if (snke\_direction\_last != SNAKE\_DIRECTION\_UP) {

if (snake\_direction\_queue.size() < 2) {

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), SNAKE\_DIRECTION\_DOWN);

}

}

break;

case sf::Keyboard::Left:

game\_paused = false;

if (snke\_direction\_last != SNAKE\_DIRECTION\_RIGHT) {

if (snake\_direction\_queue.size() < 2) {

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), SNAKE\_DIRECTION\_LEFT);

}

}

break;

case sf::Keyboard::Escape:

game\_over = true;

window.close();

cout << "You are stopped the game!" << endl;

cout << "Your score: " << score << endl << endl;

game\_over = false;

break;

case sf::Keyboard::Space:

game\_paused = true;

break;

case sf::Keyboard::Enter:

game\_paused = false;

break;

}

}

}

if (!snake\_direction\_queue.empty()) {

snake\_direction = snake\_direction\_queue.back();

snake\_direction\_queue.pop\_back();

}

if (!game\_paused) {

make\_move();

}

if (game\_over) {

sf::sleep(sf::seconds(1));

window.close();

cout << "It's GAMEOVER!" << endl;

cout << "Your score: " << score << endl << endl;

}

window.clear(sf::Color(127, 255, 212));

draw\_field(window);

window.display();

sf::sleep(sf::milliseconds(speed));

}

system("pause");

return 0;

}