МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

По Курсовой работе

по дисциплине «Программирование»

Тема: Создание игры «Змейка»

Студент гр. 9301 _____ Судаков Е.В

Преподаватель _____ Рыжов Н.Г

Санкт-Петербург
2020

22	«Змейка» - игровая программа. На экране движется змейка, состоящая из переменного числа сегментов и управляемая нажатием клавиш клавиатуры. В случайном порядке на экране разбросана пища. Если змейка «съедает» пищу, ее длина увеличивается на один сегмент, а в случайном месте экрана появляется новая пища. Игра заканчивается, если змейка натыкается сама на себя или на край игрового поля. Требуется вырастить змейку как можно большей длины. Текущая длина змейки отображается на экране.

1. Блок-схема программы(рис.1)

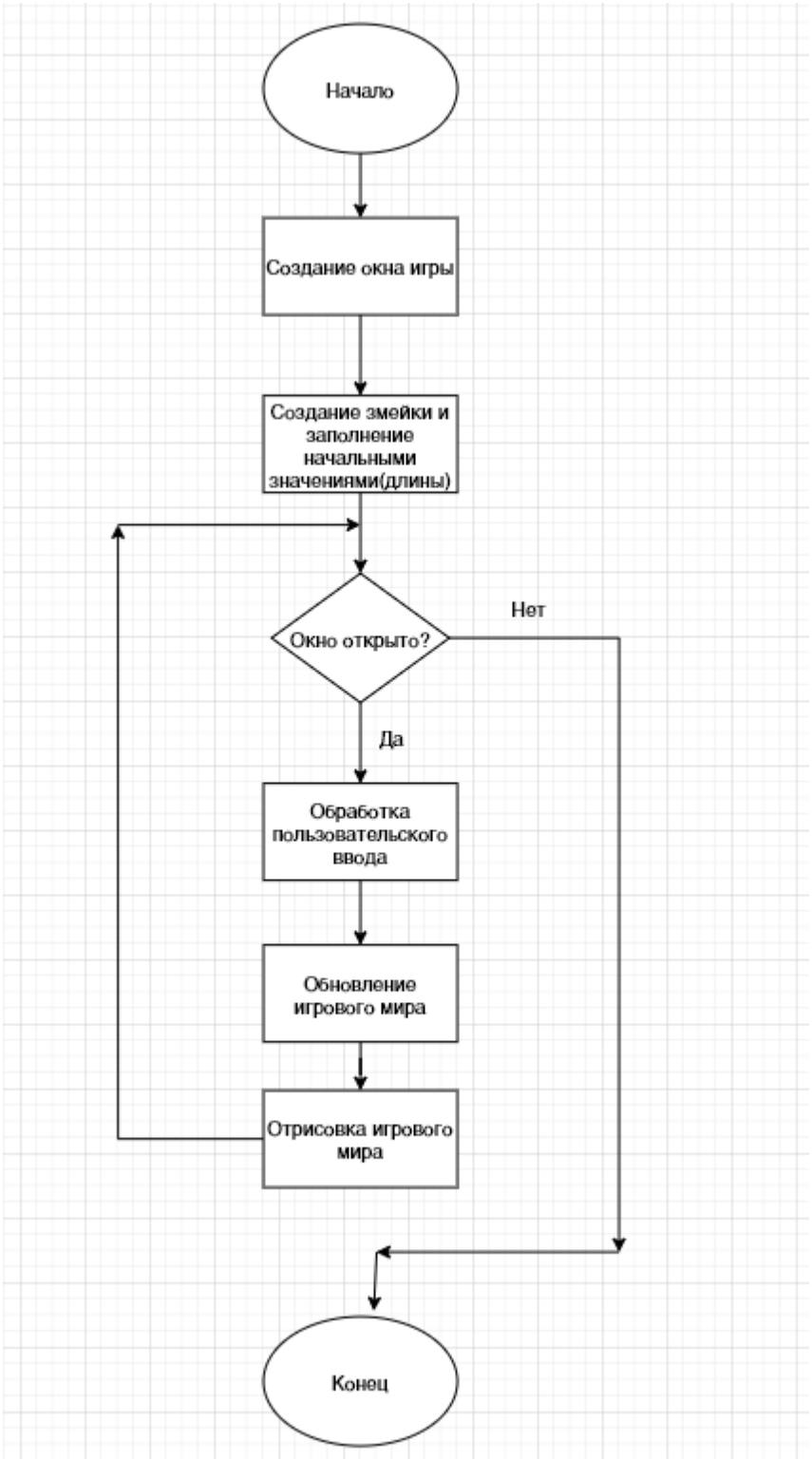

Рисунок 1. Блок схема программы

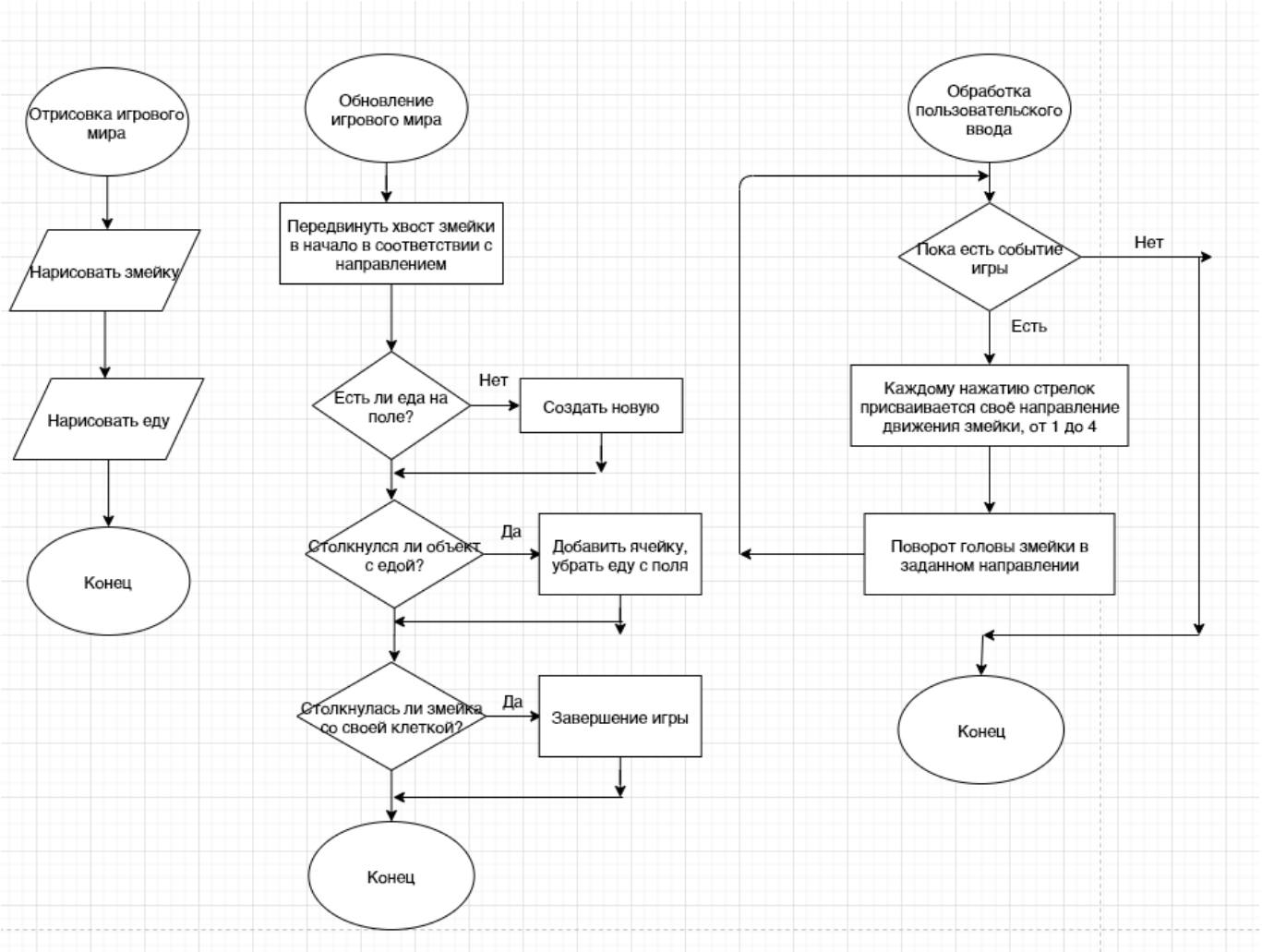

Рисунок 2 Блок схема алгоритма ввода, обновлния, отрисовки игры

2. Текст программы

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <random>            // для генерация координат еды
#include <ctime>
#include <deque>                  // для моделирование змейки
#include <windows.h>


#define fieldSize 300
#define cellSize 10
#define step 10
#define foodSize 10
#define foodIntersectionEpsilon 20   // расстояние, попав в которое еда считается съеденной
#define latency 50                   //пауза между отрисовками

std::mt19937 gen(time(0));               //Генератора рандомных чисел - Вихрь Мерсенна
std::uniform_int_distribution<> foodDistributionX(cellSize, fieldSize - cellSize);
std::uniform_int_distribution<> foodDistributionY(cellSize, fieldSize - cellSize);


enum collisionType {FOOD, CELL}; // Перечесления возможных типов столкновения змейки.
                                                        //Она может столкнуться либо с собой, либо
с едой


struct Cell {
	sf::RectangleShape rect;
	int x;
	int y;
	int direction;
};


struct Snake {
	std::deque<Cell*> body;
};


struct Food{
	sf::CircleShape shape;
	int x;
	int y;
	boolean foodOnField; // Есть ли уже еда на поле?
};



void drawCell(sf::RenderWindow &window, Cell *cell, int n) {
	//why pointer - cell* ?
	//for now i suppose there will be a lot of cells - certainly.
	//And every game loop we have to pass all that data to draw() and update() stuff
	cell->rect.setFillColor(sf::Color::Magenta);
	cell->rect.setOutlineColor(sf::Color::Black);
	cell->rect.setOutlineThickness(1);
	cell->rect.setPosition(cell->x, cell->y);
	cell->rect.setSize(sf::Vector2f(cellSize, cellSize));
	window.draw(cell->rect);
}
```

```
void drawSnake(sf::RenderWindow &window, Snake *snake) {
	/*функция отрисовки змейки. В цикле прохож по всем клеткам и вызов функции  отрисовки клетки*/
	for (int i = 0; i < snake->body.size(); i++) {
		drawCell(window, snake->body[i], i);
	}
}


void addCell(Snake *snake) {
	//добавление ячейки в конец
	Cell *push = new Cell;
	Cell *tail = snake->body[snake->body.size() - 1];
	push->x = tail->x - cellSize;
	push->y = tail->y;
	push->direction = 0;
	snake->body.push_back(push);
}


int processKeyboard(sf::Event event, sf::RenderWindow &window) {
	//Возвращает направление змейки в зависимости от ввода
	int direction = 0;
	switch (event.key.code) {
	case sf::Keyboard::Up:
		direction = 1;
		break;
	case sf::Keyboard::Down:
		direction = 2;
		break;
	case sf::Keyboard::Left:
		direction = 3;
		break;
	case sf::Keyboard::Right:
		direction = 4;
		break;
	default:
		window.close();
	}
	return direction;
}


void copyLastToFirst(Snake *snake) {
	//Все, что нужно сделать для анимации змейки - удалить хвост и добавить его к голове.
	//Поэтому структура очередь отлично подходит
	int size = snake->body.size();
	Cell *push = new Cell;
	push->direction = snake->body[0]->direction;
	push->rect = snake->body[0]->rect;
	push->x  = snake->body[0]->x;
	push->y = snake->body[0]->y;
	snake->body.push_front(push);
	snake->body.pop_back();
}


int checkYBoundary(Cell *head) {
	//check if snake is out of game field for y coords
	if (head->y > fieldSize - cellSize)
	{
		return -fieldSize;
	}
	else if (head->y < -cellSize)
```

```
	{
		return -fieldSize;
	}
	return step;
}


int checkXBoundary(Cell *head) {
	if (head->x > fieldSize - cellSize)
	{
		return -fieldSize;
	}
	else if (head->x < -cellSize)
	{
		return -fieldSize;
	}
	return step;
}


void updateSnakeCells(Snake *snake) {
	//Если змейка вышла за пределы экрана, то прибавление будет соответственно минус длинна
поля
	//Если же нет, то стандартное дельта, определенное в макросах вначале
	int deltaX = checkXBoundary(snake->body.front());
	int deltaY = checkYBoundary(snake->body.front());
	copyLastToFirst(snake);
	switch (snake->body[0]->direction) {
		case 1:
			snake->body[0]->y -= deltaY;
			break;
		case 2:
			snake->body[0]->y += deltaY;
			break;
		case 3:
			snake->body[0]->x -= deltaX;
			break;
		case 4:
			snake->body[0]->x += deltaX;
			break;
	}
}


int checkCellsRelation(Cell *c1, Cell *c2) {
	//given head of snake and 2'nd cell of snake
	//return the direction of 2'nd according to position of first

	if (c1->x == c2->x && c1->y - cellSize == c2->y) {
		return 1; //above
	}
	else if (c1->x == c2->x && c1->y + cellSize == c2->y) {
		return 2; // below
	}
	else if (c1->x - cellSize == c2->x && c1->y == c2->y) {
		return 3; // leftward
	}
	else if(c1->x + cellSize == c2->x && c1->y == c2->y){
		return 4; //right
	}
}


void rotateHead(Snake *snake, int direction) {
```

```
	//need to check if direction is possible
	int relation12 = checkCellsRelation(snake->body[0], snake->body[1]);
	int relation23 = checkCellsRelation(snake->body[1], snake->body[2]); // check if right or
left rotation is even possible
	// because maybe there some 3'd cell;
	int x = snake->body.front()->x;
	int y = snake->body.front()->y;
	if (x <= 0 || x >= fieldSize || y <= 0 || y >= fieldSize) {
		//while head is out field not allowed to rotate
		return;
	}
	if (direction == 1 && relation12 != 1) //up
	{
		snake->body[0]->direction = direction;
	}
	else if (direction == 2 && relation12 != 2) //down
	{
		snake->body[0]->direction = direction;
	}
	else if (direction == 3 && relation12 != 3 && relation23 != 3) // left
	{
		snake->body[0]->direction = direction;
	}
	else if (direction == 4 && relation12 != 4 && relation23 != 4) // right
	{
		snake->body[0]->direction = direction;
	}
}


//@utility
void fillSnake(Snake *snake, int n) {
	//Вспомогательная функция для создания начальной змейки
	for (int i = 0; i < n; i++) {
		addCell(snake);
	}
}


void drawFood(Food *piece, sf::RenderWindow &window) {
	//Отрисовка еды
	piece->shape.setPosition(sf::Vector2f(piece->x, piece->y));
	piece->shape.setFillColor(sf::Color::Red);
	piece->shape.setRadius(foodSize);
	window.draw(piece->shape);
}



void generateFood(Food *food ,int x, int y) {
	//Создание новой еды.
	std::cout << " X :: " << x << " Y :: " << y;
	food->x = x;
	food->y = y;
}


int getDistance(int x1, int y1, int x2, int y2) {
	//Вычислени расстояние меджу двумя точками
	double dx = x1 - x2;
	double dy = y1 - y2;
	return sqrt(dx * dx + dy * dy);
}
```

```
int collisionDetection(Cell *head, int x, int y, int r, collisionType t) {
	// x, y, r - params of entity to check with
	// if checking with other cell, r just = 0

	double l = getDistance(head->x, head->y, x+r, y+r);
	switch (t) {
		case FOOD :
			if (l < foodIntersectionEpsilon) {
				std::cout << "collision with food\n";
				return 1;
			}
			break;
		case CELL:
			if (l == 0) {
				std::cout << "collision with cell\n";
				return 1;
			}
	}
	return 0;
}


void updateFood(Food *food) {
	//Как только еды не оказалось на поле, генерируется новая
	int x, y;
	if (!(food->foodOnField)) {
		x = foodDistributionX(gen);
		y = foodDistributionY(gen);
		generateFood(food, x, y);
		food->foodOnField = true;
	}
}


void updateGameWorld(Snake *snake, Food *food) {
	/*  Обновление игрового мира
		Вызов функции перемещения клеток змейки и функции обновления состояния еды.
		Так же вычисляется, произошло ли столкновение с клеткой или едой
	*/
	std::cout << "\r" << snake->body[0]->x << " " << snake->body[0]->y;
	updateSnakeCells(snake);
	updateFood(food);
	collisionType f = FOOD;
	collisionType c = CELL;
	if (collisionDetection(snake->body[0], food->x, food->y, food->shape.getRadius(), f)) {
		addCell(snake);
		food->foodOnField = false;
	}
	for (int i = 1; i < snake->body.size(); i++) {
		if (collisionDetection(snake->body[0], snake->body[i]->x, snake->body[i]->y, 0, c))
{
			exit(-1);
		}
	}
}



void proccesGameInput(sf::RenderWindow &window, Snake *snake, sf::Event event) {
	//Стандартная обработка пользовательского ввода
	//Если введена клавиша - стрелка, нужно развернуть голову змейки в номом, заданном
направлении
	int direction;
```

```
	while (window.pollEvent(event)) {
		if (event.type == sf::Event::Closed)
			window.close();
		if (event.type == sf::Event::KeyPressed) {
			direction = processKeyboard(event, window);
			if (snake->body[0]->direction != direction) {
				rotateHead(snake, direction);
			}
		}
	}
}



void drawGame(Snake *snake, Food *food, sf::RenderWindow &window) {
	//Отрисовки змейки и еды
	drawSnake(window, snake);
	drawFood(food, window);
	window.display();
	window.clear();
	sleep(sf::milliseconds(latency));
}


int main()
{
	sf::RenderWindow window(sf::VideoMode(fieldSize, fieldSize), "Snakie!");
	sf::Event event;
	window.pollEvent(event);


	Cell *cell1 = new Cell;
	cell1->x = 100;
	cell1->y = 100;
	cell1->direction = 4;

	Snake *snake = new Snake; //again, we don't know how much of stack available here. Using
heap therefore.
	snake->body = std::deque<Cell*>();
	snake->body.push_back(cell1);

	fillSnake(snake, 5);

	drawSnake(window, snake);

	Food *food = new Food;
	food->foodOnField = false;

	int direction = 0;

	while (window.isOpen()) {
		//Игровой цикл
		proccesGameInput(window, snake, event);
		updateGameWorld(snake, food);
		drawGame(snake, food, window);
	}

	return 0;
}
```

3. Описание програмы

Программа предлагает пользователю провести время за классической игрой змейка. В программе змейка представлена как очередь – ведь для движения змейки необходимо менят координаты только «головы» и «хвоста», остальные во время одного хода остаются неизменными

4. Руководство пользователя

(1) Назначение программы: Программа носит развлекательный характер, предлагая пользователю сыграть в «змейку»

(2) Условия применения: Распространяется под лицензией MIT. Ограничения :x86-64 совместимый процессор вычислительной машины.

(3) Пуск программы : Для запуска программы дважды нажмите на исполняемый файл (Snakie!.exe).

(4) Сообщения оператору: вывод рабочей сессии (рис.5)

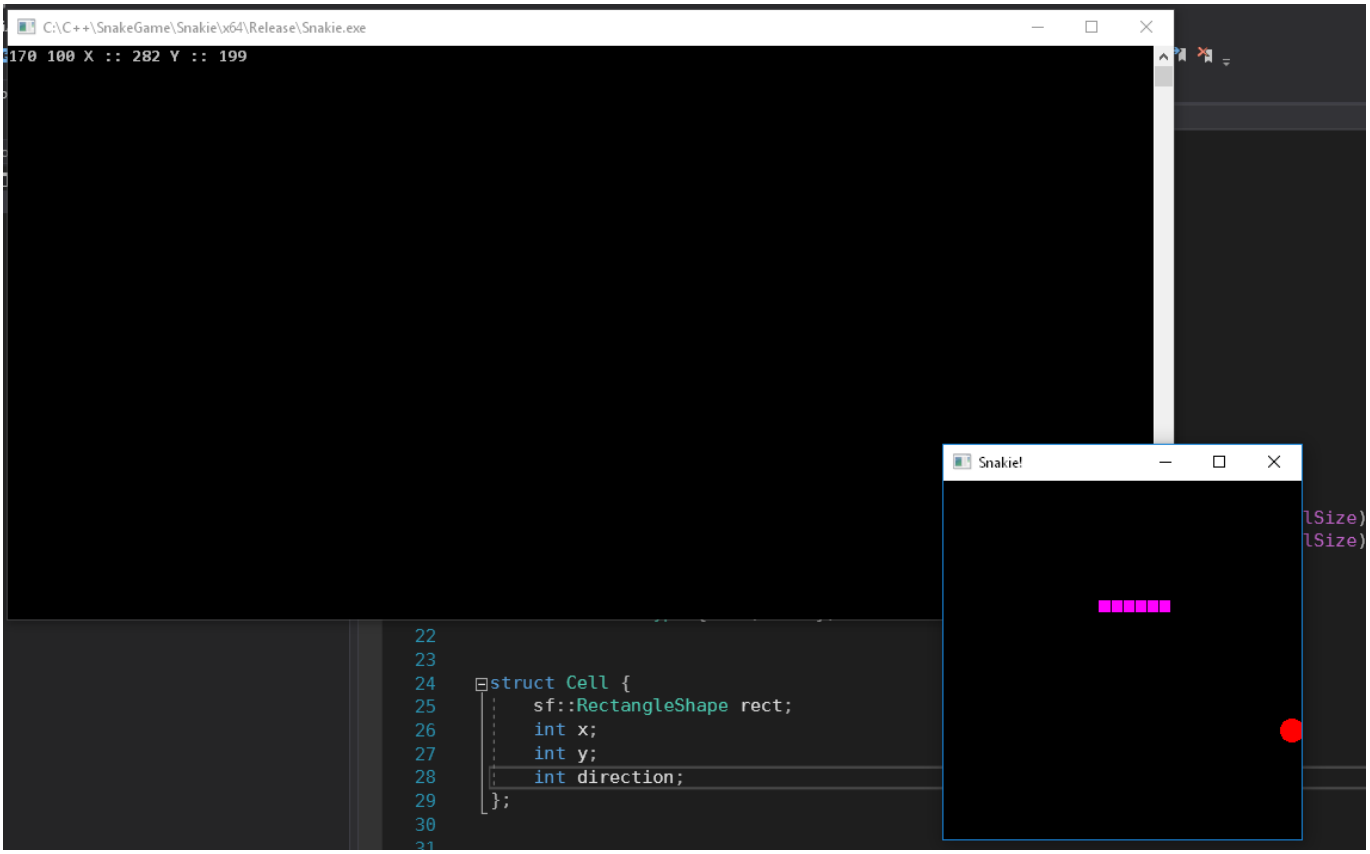

Рис 2. Игра в начале

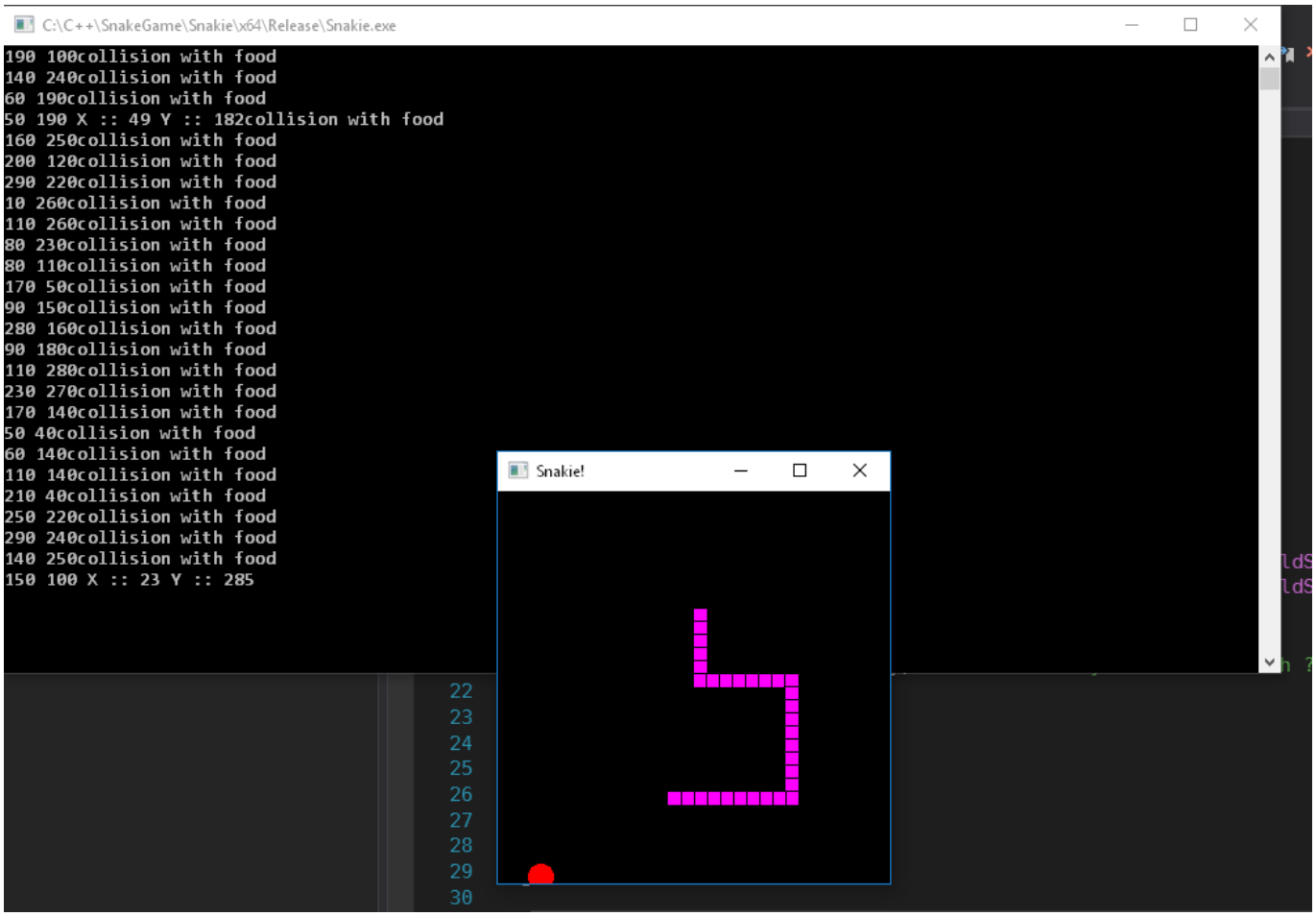

Рисунок 3 Игра после некоторого времени

5. Пути дальнейшего улучшения программы

-Перенести змейку в 3D

- Добавить звуковые эффекты