Министерство образования и науки Российской Федерации Федеральное агентство по образованию

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Разработка приложение на основе ООП парадигмы

Отчет по лабораторной работе №6 дисциплины

«Технологии программирования»

Выполнил студент группы ИВТ-22 /Крючков И. С/ Проверил /Долженкова М. Л./

Киров 2022

1. Задание

В выбранной предметной области создать иерархию классов состоящую минимум из одного родительского и двух дочерних классов. В каждом классе определить минимум два собственных член данных, две собственных, две унаследованных и две перекрытых член функции. Разработать приложение демонстрирующее принципы полиморфизма, наследования и инкапсуляции.

1. Иерархия классов



Рисунок 1 – Иерархия классов

1. Экранные формы

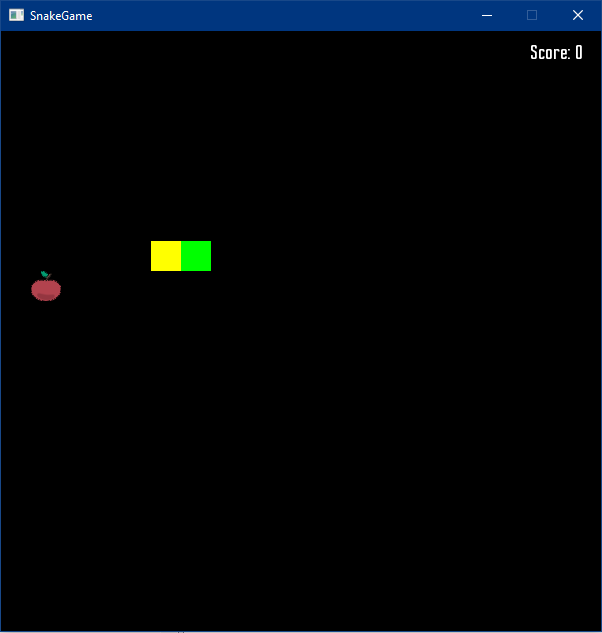


Рисунок 2 – Главное окно программы



Рисунок 2 – Проигрыш

1. Листинг кода

**feed.h**

#include <SFML/Graphics.hpp>

class Feed {

public:

Feed(sf::RenderWindow \*, int);

Feed() = default;

void set\_feed(int \*\*&, int, int, int);

virtual void set\_scope(int);

virtual int get\_scope();

void drawFeed();

sf::RenderWindow \*window;

int fldsize;

sf::Texture txtr;

sf::Sprite sprt;

private:

int scope;

};

class Accelerator : public Feed {

public:

Accelerator(sf::RenderWindow \*, int);

Accelerator() = default;

// перекрытые

void set\_scope(int);

int get\_scope();

//унаследованные

// void drawFeed();

// void set\_feed(int \*\*&, int, int, int);

// собственные

int getBoost();

void setBoost();

private:

int boost;

int bonusScope;

};

class Grib : public Feed {

public:

Grib(sf::RenderWindow \*, int);

Grib() = default;

// перекрытые

void set\_scope(int);

int get\_scope();

// унаследованные

// void drawFeed();

// void set\_feed(int \*\*&, int, int, int);

// собственные

void setDamage();

int getDamage();

private:

int damage;

int fineScope;

};

**feed.cpp**

#include "feed.h"

#include <iostream>

Feed::Feed(sf::RenderWindow \*w, int field\_size){

window = w;

fldsize = field\_size;

scope = 50;

if(!txtr.loadFromFile("./img/apple.png")){

std::cout << "img apple.png load error" << std::endl;

}

sprt.setTexture(txtr);

}

void Feed::set\_feed(int \*\*&field, int cell\_size, int field\_size, int type\_feed){

struct position

{

int x;

int y;

};

position rand\_coords;

std::vector<position> coords;

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

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

if(field[i][j] == 0){

coords.push\_back({i, j});

}

}

}

rand\_coords = coords[std::rand() % (coords.size()-1)];

field[rand\_coords.x][rand\_coords.y] = type\_feed;

sprt.setPosition(rand\_coords.x\*cell\_size, rand\_coords.y\*cell\_size);

}

void Feed::drawFeed(){

window->draw(sprt);

}

void Feed::set\_scope(int sc){

scope = sc;

}

int Feed::get\_scope(){

return scope;

}

Accelerator::Accelerator(sf::RenderWindow \*w, int field\_size){

window = w;

fldsize = field\_size;

bonusScope = 200;

if(!txtr.loadFromFile("./img/accelerator.png")){

std::cout << "img accelerator.png load error" << std::endl;

}

sprt.setTexture(txtr);

}

void Accelerator::set\_scope(int sc){

bonusScope = sc;

}

int Accelerator::get\_scope(){

return bonusScope;

}

void Accelerator::setBoost(){

int b[3] = { 100, 150, 200 };

boost = b[std::rand() % 3];

}

int Accelerator::getBoost(){

return boost;

}

Grib::Grib(sf::RenderWindow \*w, int field\_size){

window = w;

fldsize = field\_size;

fineScope = -200;

if(!txtr.loadFromFile("./img/grib.png")){

std::cout << "img grib.png load error" << std::endl;

}

sprt.setTexture(txtr);

}

void Grib::set\_scope(int sc){

fineScope = sc;

}

int Grib::get\_scope(){

return fineScope;

}

void Grib::setDamage(){

int d[3] = { 1, 3, 5 };

damage = d[std::rand() % 3];

}

int Grib::getDamage(){

return damage;

}

**snake.h**

#include <SFML/Graphics.hpp>

class Snake{

public:

Snake(sf::RenderWindow \*, int, int, int \*\*&);

int moveSnake(int, int \*\*&);

void drawSnake();

void setTurnSnake(int);

int getDirection();

void dec\_snake\_length(int, int, int \*\*&);

void resetSnake(int, int \*\*&);

private:

sf::RenderWindow \*window;

int turn\_direction; // направление поворота

int direction;

std::vector<sf::RectangleShape> snake\_body;

struct position {

int x;

int y;

};

std::vector<position> snake\_pos;

sf::Color colorHead;

sf::Color colorBody;

int snake\_length;

int fldsize;

position last\_pos\_tail;

sf::RectangleShape rectangle;

void inc\_snake\_length(int, int, int, sf::Color, int \*\*&);

int checkIntersections(int, int, int \*\*&);

};

**snake.cpp**

#include "snake.h"

#include <iostream>

#include <algorithm>

Snake::Snake(sf::RenderWindow \*w, int cell\_size, int field\_size, int \*\*&field){

int x\_pos, y\_pos, x\_pos\_window, y\_pos\_window;

fldsize = field\_size;

window = w;

snake\_length = 0;

// направление движения при повороте

// (0-nothing; 1-U; 2-R; 3-D; 4-L)

turn\_direction = 0;

direction = 2; // текущее направление движения

colorHead = sf::Color::Yellow;

colorBody = sf::Color::Green;

rectangle = sf::RectangleShape { sf::Vector2f(cell\_size, cell\_size) };

// голова

x\_pos = fldsize/2;

y\_pos = fldsize/2;

inc\_snake\_length(x\_pos, y\_pos, cell\_size, colorHead, field);

//ячейка тела

x\_pos = fldsize/2 - 1;

y\_pos = fldsize/2;

inc\_snake\_length(x\_pos, y\_pos, cell\_size, colorBody, field);

last\_pos\_tail = {x\_pos-1, y\_pos};

}

void Snake::drawSnake(){

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

window->draw(snake\_body[i]);

}

}

void Snake::inc\_snake\_length(int x\_pos, int y\_pos, int cell\_size, sf::Color color, int \*\*&field){

int x\_pos\_window, y\_pos\_window;

x\_pos\_window = x\_pos \* cell\_size;

y\_pos\_window = y\_pos \* cell\_size;

rectangle.setPosition(x\_pos\_window, y\_pos\_window);

rectangle.setFillColor(color);

snake\_body.push\_back(rectangle);

snake\_pos.push\_back({x\_pos, y\_pos});

field[x\_pos][y\_pos] = 1;

snake\_length++;

}

void Snake::dec\_snake\_length(int damage, int cell\_size, int \*\*&field){

int del\_num, x\_pos, y\_pos, i\_del\_num;

i\_del\_num = snake\_length - 2;

del\_num = std::min(i\_del\_num, damage);

for(int i = 0; i < del\_num; i++){

snake\_body.pop\_back();

x\_pos = snake\_pos.back().x;

y\_pos = snake\_pos.back().y;

field[x\_pos][y\_pos] = 0;

snake\_pos.pop\_back();

snake\_length--;

}

}

int Snake::checkIntersections(int dx, int dy, int \*\*&field){

int spx, spy;

// пересечения с границами поля

spx = snake\_pos[0].x + dx;

spy = snake\_pos[0].y + dy;

if(spx >= fldsize || spx < 0){

return 1;

}else if(spy >= fldsize || spy < 0){

return 1;

}

if(field[spx][spy] != 0){

// пересечение с телом змеи

if(field[spx][spy] == 1){

return 1;

// пересечение с яблоком

}else if(field[spx][spy] == 2){

field[spx][spy] = 0;

return 2;

// пересечение с молнией

}else if(field[spx][spy] == 3){

field[spx][spy] = 0;

return 3;

// пересечение с грибом

}else{

field[spx][spy] = 0;

return 4;

}

}

return 0;

}

int Snake::moveSnake(int cell\_size, int \*\*&field){

int x\_pos, y\_pos, x\_pos\_window, y\_pos\_window;

int dx, dy, tmp\_x, tmp\_y;

int current\_direction, intersected;

int return\_val = 0;

direction = (turn\_direction != 0) ? turn\_direction : direction;

turn\_direction = 0;

switch (direction) {

case 1: //up

dx = 0;

dy = -1;

break;

case 2: //right

dx = 1;

dy = 0;

break;

case 3: // down

dx = 0;

dy = 1;

break;

case 4: //left

dx = -1;

dy = 0;

break;

}

intersected = checkIntersections(dx, dy, field);

//проверка на пересечение

if(intersected != 0){

// со стеной

if (intersected == 1){

return 1;

// с грибом

}else if(intersected == 4){

return 4;

// с молнией или яблоком

}else{

// увеличение змеи

x\_pos = last\_pos\_tail.x;

y\_pos = last\_pos\_tail.y;

inc\_snake\_length(x\_pos, y\_pos, cell\_size, colorBody, field);

return\_val = intersected;

}

}

// перемещение хвоста

tmp\_x = snake\_pos[snake\_length-1].x;

tmp\_y = snake\_pos[snake\_length-1].y;

field[tmp\_x][tmp\_y] = 0;

last\_pos\_tail = {tmp\_x, tmp\_y};

//перемещение тела

for(int i = snake\_length-1; i > 0; i--){

x\_pos\_window = snake\_body[i-1].getPosition().x;

y\_pos\_window = snake\_body[i-1].getPosition().y;

snake\_body[i].setPosition(x\_pos\_window, y\_pos\_window);

snake\_pos[i] = snake\_pos[i-1];

}

// перемещение головы

x\_pos\_window = snake\_body[0].getPosition().x + dx\*cell\_size;

y\_pos\_window = snake\_body[0].getPosition().y + dy\*cell\_size;

snake\_body[0].setPosition(x\_pos\_window, y\_pos\_window);

snake\_pos[0].x += dx;

snake\_pos[0].y += dy;

field[snake\_pos[0].x][snake\_pos[0].y] = 1;

return return\_val;

}

void Snake::setTurnSnake(int turn){

turn\_direction = turn;

}

int Snake::getDirection(){

return direction;

}

void Snake::resetSnake(int cell\_size, int\*\*& field){

int x\_pos, y\_pos;

direction = 2;

turn\_direction = 0;

snake\_body.clear();

snake\_pos.clear();

snake\_length = 0;

// голова

x\_pos = fldsize/2;

y\_pos = fldsize/2;

inc\_snake\_length(x\_pos, y\_pos, cell\_size, colorHead, field);

//ячейка тела

x\_pos = fldsize/2 - 1;

y\_pos = fldsize/2;

inc\_snake\_length(x\_pos, y\_pos, cell\_size, colorBody, field);

last\_pos\_tail = {x\_pos-1, y\_pos};

}

**main.cpp**

#include <SFML/Graphics.hpp>

#include <iostream>

#include "snake.h"

#include "feed.h"

#include <ctime>

const int def\_speed = 300; //ms

const int field\_size = 20;

const int cell\_size = 30;

bool game\_started = false;

int speed = def\_speed;

int speed\_times = 5;

int \*\*field;

int scope = 0;

int wnd\_width = 600;

int wnd\_height = 600;

bool game\_over = false;

bool start\_menu = true;

sf::Font font;

sf::Text text\_score;

sf::Text text\_gameover;

sf::Text text\_start;

void draw\_scope(sf::RenderWindow &window, int sc){

text\_score.setString("Score: " + std::to\_string(sc));

text\_score.setPosition(wnd\_width - text\_score.getLocalBounds().width - 20, 10);

window.draw(text\_score);

}

int main()

{

std::srand(time(0));

sf::RenderWindow window(sf::VideoMode(wnd\_width, wnd\_height), "SnakeGame", sf::Style::Close | sf::Style::Titlebar);

window.setVerticalSyncEnabled(true);

window.setFramerateLimit(60);

font.loadFromFile("font/Ampero-Regular.ttf");

text\_score.setFont(font);

text\_score.setCharacterSize(18);

text\_score.setFillColor(sf::Color::White);

text\_gameover.setFont(font);

text\_gameover.setCharacterSize(32);

text\_gameover.setFillColor(sf::Color::White);

text\_gameover.setString("Game Over (r - restart)");

text\_gameover.setPosition((wnd\_width - text\_gameover.getLocalBounds().width) / 2, (wnd\_height - text\_gameover.getLocalBounds().height) / 2);

text\_start.setFont(font);

text\_start.setCharacterSize(32);

text\_start.setFillColor(sf::Color::White);

text\_start.setString("Press <Space> to start");

text\_start.setPosition((wnd\_width - text\_start.getLocalBounds().width) / 2, (wnd\_height - text\_start.getLocalBounds().height) / 2);

field = new int\* [field\_size];

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

field[i] = new int [field\_size];

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

field[i][j] = 0;

}

}

Snake\* snake = new Snake(&window, cell\_size, field\_size, field);

Accelerator\* accelerator = new Accelerator(&window, field\_size);

Grib\* grib = new Grib(&window, field\_size);

Feed\* feed = new Feed(&window, field\_size);

Feed\*\* mas = new Feed\* [3];

mas[0] = feed;

mas[1] = accelerator;

mas[2] = grib;

std::vector<int> snake\_direction\_queue;

int last\_snake\_direction, move\_snake\_result;

int dmg;

int r = 0;

feed->set\_feed(field, cell\_size, field\_size, 2);

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

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

last\_snake\_direction = !snake\_direction\_queue.empty() ? snake\_direction\_queue[0] : snake->getDirection();

switch (event.key.code) {

case sf::Keyboard::Up:

if(last\_snake\_direction != 3 && last\_snake\_direction != 1){

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

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), 1);

}

}

break;

case sf::Keyboard::Right:

if(last\_snake\_direction != 4 && last\_snake\_direction != 2){

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

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), 2);

}

}

break;

case sf::Keyboard::Down:

if(last\_snake\_direction != 1 && last\_snake\_direction != 3){

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

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), 3);

}

}

break;

case sf::Keyboard::Left:

if(last\_snake\_direction != 2 && last\_snake\_direction != 4){

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

snake\_direction\_queue.insert(snake\_direction\_queue.begin(), 4);

}

}

break;

case sf::Keyboard::R:

if(game\_over){

scope = 0;

speed = def\_speed;

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

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

field[i][j] = 0;

}

}

snake->resetSnake(cell\_size, field);

feed->set\_feed(field, cell\_size, field\_size, 2);

r = 0;

game\_started = true;

game\_over = false;

}

break;

case sf::Keyboard::Space:

if(!game\_started){

game\_started = true;

}

break;

}

}

}

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

snake->setTurnSnake(snake\_direction\_queue.back());

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

}

if(start\_menu){

window.clear();

window.draw(text\_start);

start\_menu = false;

}

if (game\_started){

window.clear();

draw\_scope(window, scope);

move\_snake\_result = snake->moveSnake(cell\_size, field);

if(move\_snake\_result != 0){

// столкновение

if(move\_snake\_result == 1){

game\_started = false;

game\_over = true;

window.draw(text\_gameover);

// еда

}else{

// ускорение

if(move\_snake\_result == 3){

scope += mas[1]->get\_scope();

Accelerator\* rrr = (Accelerator\*)mas[1];

speed = rrr->getBoost();

speed\_times = 3;

// гриб

}else if (move\_snake\_result == 4){

scope = std::max(scope+grib->get\_scope(), 0);

dmg = grib->getDamage();

snake->dec\_snake\_length(dmg, cell\_size, field);

// яблоко

}else{

scope += feed->get\_scope();

speed\_times--;

if(speed\_times == 0){

speed = def\_speed;

}

}

r = std::rand() % 3;

Accelerator\* rrr = (Accelerator\*)mas[1];

Grib\* ttt = (Grib\*)mas[2];

switch (r)

{

case 0:

feed->set\_feed(field, cell\_size, field\_size, 2);

break;

case 1:

mas[1]->set\_feed(field, cell\_size, field\_size, 3);

rrr->setBoost();

break;

case 2:

mas[2]->set\_feed(field, cell\_size, field\_size, 4);

ttt->setDamage();

break;

}

}

}

snake->drawSnake();

switch (r)

{

case 0:

feed->drawFeed();

break;

case 1:

mas[1]->drawFeed();

break;

case 2:

mas[2]->drawFeed();

break;

}

}

window.display();

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

}

return 0;

}

1. Вывод

В ходе выполнения лабораторной работы была написана игра «змейка». Для реализации необходимого функционала были разработаны три класса: один родительский и два дочерних. Каждый класс имеет минимум два собственных член данных, две собственных, две унаследованных и две перекрытых член функции. В структуре классов применяются принципы инкапсуляции, наследования и полиморфизма.