Министерство науки и высшего образования РФ

ФГАОУ ВО ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИУ)

Институт естественных и точных наук

Факультет математики, механики и компьютерных технологий

Кафедра прикладной математики и программирования

«Графическая оконная библиотека»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ   
по дисциплине «Объектно-ориентированное программирование»

ЮУрГУ–01.03.02.2021.101.ПЗ КР

|  |  |  |
| --- | --- | --- |
|  |  | *Руководитель,*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Демидов А.К.*  *«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023г.* |
|  |  | *Автор работы:*  *Студент группы: ЕТ – 212*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Никитин М.С.*  *«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023г.* |
|  |  | *Работа защищена с оценкой*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г.* |

Челябинск 2023

Федеральное государственное автономное образовательное учреждение высшего образования

«Южно-Уральский государственный университет   
(национальный исследовательский университет)»

Институт естественных и точных наук

Кафедра «Прикладная математика и программирование»

Направление Прикладная математика и информатика

УТВЕРЖДАЮ

Заведующий кафедрой ПМиП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А.Замышляева

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

**ЗАДАНИЕ**

**на курсовую работу студента**

\_\_\_\_\_\_\_\_*Никитин М.С.*\_\_\_\_\_\_\_\_\_

Группа \_\_*ЕТ-212*\_\_\_\_

1. Дисциплина  *Объектно-ориентированное программирование*

2. Тема работы \_\_\_\_*Программа для обучения основам программирования* \_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Срок сдачи студентом законченной работы *28 января 2023 г.*

4. Перечень вопросов, подлежащих разработке

1. разработка иерархии и интерфейса классов;
2. реализация программы (библиотеки классов) на языке С++
3. оформление программной документации (описание программы (библиотеки классов), руководство пользователя, листинг кода) и отчета по курсовой работе
4. презентация проектных решений для защиты КР (иерархия и интерфейсы классов, особенности реализации)

5. Календарный план

|  |  |  |
| --- | --- | --- |
| **Наименование разделов**  **(этапов) курсовой работы** | **Срок выполнения**  **разделов (этапов) работы** | **Отметка**  **о выполнении**  **руководителя** |
| анализ предметной области | 01.09.2022-10.10.2022 |  |
| разработка иерархии и интерфейса классов | 20.09.2022-07.11.2022 |  |
| реализация основных классов, функций | 01.10.2022-20.11.2022 |  |
| тестирование программы и/или классов, улучшение и исправление ошибок | 20.10.2022-10.12.2022 |  |
| оформление программной документации и отчета по курсовой работе | 30.10.2022-20.12.2022 |  |
| защита курсовой работы | 20.12.2022-28.12.22 |  |

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

АННОТАЦИЯ

Никитин М.С. Программа для обучения основам программирования. – Челябинск: ЮУрГУ, ЕТ-212, 2023. – 30с., 3 ил., библиографический список – 3 наим., 1 прил.

В курсовой работе описывается разработка программы для обучения основам программирования с помощью объектно-ориентированного подхода. Работа содержит результаты объектно-ориентированного анализа и проектирования, инструкции по установке и использованию библиотеки.

В результате работы была разработана графическая оконная библиотека, код которой приводится в приложении.

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 5](#_Toc125587768)

[1 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc125587769)

[**2 ОПИСАНИЕ ПРОГРАММЫ** 6](#_Toc125587770)

[**3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ** 11](#_Toc125587771)

[**4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ** 11](#_Toc125587772)

[ЗАКЛЮЧЕНИЕ 13](#_Toc125587773)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 14](#_Toc125587774)

[ПРИЛОЖЕНИЕ А 15](#_Toc125587775)

# ВВЕДЕНИЕ

**Актуальность темы**. Объектно-ориентированный подход является наиболее прогрессивной технологией разработки программных систем, позволяет разрабатывать более сложные системы.

**Цель работы** – разработать программу для обучения основам программирования.

**Задачи работы**:

– изучить приемы объектно-ориентированного анализа;

– научиться разрабатывать программы в объектно-ориентированном стиле;

– овладеть технологиями объектно-ориентированного анализа и проектирования;

– изучить особенности объектной модели языка программирования C++;

– научиться самостоятельно и творчески использовать знания и полученные практические навыки;

– овладеть навыками самостоятельного получения новых знаний по теории и практике объектного подхода в программировании.

**Объект работы** – программа для обучения основам программирования.

**Предмет работы** – применение объектно-ориентированного подхода для разработки библиотеки.

**Результаты работы** можно использовать в процессе последующего обучения в соответствии с учебным планом подготовки бакалавров по направлению «Прикладная математика и информатика».

# 1 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать программу для обучения основам программирования «урожай» со следующими возможностями:

* генерация прямоугольного поля фиксированного размера;
* генерация и урожая на случайных клетках поля;
* генерация роботов, стрелок, банок с краской и выхода на клетках поля, определённых фалом с заданием;
* выбор пользователем желаемого набора программ;
* перемещение пользователем незафиксированных программ;
* перемещение пользователем незафиксированных роботов;
* проверка на столкновение нескольких роботов;
* запуск разработанной пользователем программы.

Анализ предметной области выявляет следующие объекты:

* клетчатое рабочее поле размером m × n;
* объекты, фиксированные на поле;
* роботы, отличающиеся цветом, имеющие направление и координаты;
* команды, содержащие возможные действия с объектами;
* программы, представляющие собой команды определённых координат и цвета;
* задания с описанием, с помощью которых настраиваются роботы и команды.

**2 ОПИСАНИЕ ПРОГРАММЫ**

2.1 Для разработки программы были использованы:

– компилятор MinGW C/C++ 11.3;

– библиотека winBGIm.

2.2 Программа состоит из 2 модулей:

1 Модуль main (файл main.cpp) содержит следующие функции:

void clearWin(); //очистка окна

void put\_text(string text\_task); //вставка текста задания

void drawBarCurrectProg(int i); //отрисовка индикатора выбранной программы в нижней панели

void reDraw(string text\_task); // перерисовка всех элементов во время выполнения программы роботами

void highlightCell(position current\_cell); // выделение ячейки

void drawCurrectProg(int n\_currect\_com, position current\_cell); //отрисовка текущей программы

int main();

2 Модуль **Objects** (интерфейсная часть в файле Objects.h, реализация в файле Objects.cpp) содержит следующие классы:

//структура для хранения координат парами

struct position {

int x, y; // координаты

position() {x = y = 0;} // конструктор без передаваемых параметров

position(int new\_x, int new\_y):x(new\_x), y(new\_y) {}

position(const position &pos) {x = pos.x;y = pos.y;} // конструктор копий

// перегрузка оператора для сравнения

friend auto operator<=>(const position&, const position&) = default;

};

class Robot {

IMAGE \*img; // картинка робота

position pos; //текущие координаты робота;

position direction; // текущее направление (смещение по координатам) {0, 1}/{1, 0}/{0, -1}/{-1, 0}

int color; // текущий цвет

bool allow\_change\_direction; // разрешено изменять направление?

bool allow\_change\_cordinat; // разрешено изменять координаты?

public:

Robot(IMAGE \*img\_robot,bool is\_allow\_change\_direction, bool is\_allow\_change\_cordinat); //конструктор

~Robot(); //деструктор

void set\_cordinat(position); // установить координаты

void set\_direction(position); // установить направление

void set\_color(int); // установить цвет

int get\_color(); // вернуть цвет

position get\_cordinat(); // вернуть координаты

position get\_direction(); // вернуть направление

void change\_Field(Field &); // перед выходом из клетки удаление или замена объекта

void draw(); // рисование

private:

bool is\_crash(vector <Robot \*> &Robots); // столкнулся(набор роботов)?

};

//базовый класс для неподвижных сущностей

class Object {

protected:

IMAGE \*img; // картинка объекта

public:

Object(IMAGE \*); // конструктор

Object(const Object &obj); // Конструктор копий

~Object();// деструтор

virtual void draw(position) = 0; // вернуть картин

virtual bool is\_access(Robot &) = 0; // проверка клетки на доступность для робота

};

// объект доступный для сбора

class Fruit : public Object {

public:

Fruit(IMAGE \*);// конструктор

~Fruit(); // деструктор

void draw(position); // рисование в позиции

bool is\_access(Robot &); // проверка на доступность

};

// объект не доступный для перемещения

class Tree : public Object {

public:

Tree(IMAGE \*); //конструктор

~Tree(); // деструктор

void draw(position); // рисование в позиции

bool is\_access(Robot &); // проверка на доступность

};

// струкура для клетки поля

struct Cell {

Object \*current\_object = nullptr; // объект в клетке

int color = WHITE; // цвет клетки

};

//класс поля

class Field {

int width, height; // размеры поля

vector<vector<Cell>> fullField; //поле из клеток

public:

Field(int, int); // конструктор

void set\_obj(Object \*, position); // установить объект

void delete\_obj(position); // удалить объект

void set\_color(position, int); // установить цвет

void draw(); // рисование всех Object на поле

Object \*get\_object(position); // получить объект на клетке

};

class Command {

bool is\_allow\_change\_cordinat; //разрешено изменять координаты?

bool is\_allow\_delete; //разрешено удалять?

friend class Programm; //указываем, что Programm – дружественный класс

protected:

IMAGE \*img; // изображение команды

position coord; //координаты команды

public:

Command(bool, bool, position); //конструктор

Command(const Command &com); // Конструктор копий

Command() = default; // Конструктор копий

void set\_pos(position); // задать новую позицию команде

position get\_pos(); // вернуть текущую позицию

virtual void use(Robot &) = 0; // виртуальный метод на воздействие на робота

virtual void draw(int color) = 0; // виртуальный метод рисования

};

// "Стрелка", меняющая направление

class Arrow : public Command {

position orientation; // ориентация стрелки

public:

// разрешение на перемещения и удаление, позиция расположения, позимещения изменения направления

Arrow(bool is\_allow\_change\_cordinat, bool is\_allow\_delete, position coord, position orient); // конструктор

void use(Robot &); // применить к роботу

void draw(int color); // отобразить

};

//"Банка с краской", меняющая цвет робота

class ChangeColor : public Command {

int color; // цвет для перекраски

public:

ChangeColor(bool is\_allow\_change\_cordinat, bool is\_allow\_delete, position, int color); // конструктор

void use(Robot &); // применить к роботу

void draw(int color); // отобразить

};

//"Выход", удаляющий робота

class Exit : public Command {

public:

Exit(bool is\_allow\_change\_cordinat, bool is\_allow\_delete, position); // конструктор

void use(Robot &);// применить к роботу

void draw(int color); // отобразить

};

// Класс программ, содержащий набор комманд одного цвета

class Programm {

int color = WHITE; // цвет программы

vector<Command \*> commands ; // вектор комманд внутри одной программы

public:

Programm(int color); // конструктор

int get\_col(); // вернуть цвет

void add(Command \*);// добавление комманды

void draw(); // отрисовать все комманды данной программы

Command \*select(position); // выбрать команду в position

void delete\_com(Command\*); // удаление комманды

};

extern vector <Robot\*> Robots; // глобальный вектор с роботами

extern vector <Programm\*> Programms; // глобальный вектор с программами

class Task {

string text\_task; // текст задания

string name\_taskFile; // название файла с заданием

int count\_robots; // количество роботов

int count\_commands;// количество комманд

int count\_tree; // количество деревьев

int count\_fruit; // количество урожая

// инициализация всех компанентов согласно заданию

public:

Task(const string);// название файла с заданием

// инициализация робтов и программ по файлу с заданием

void initialize(Field &, std::vector <Robot \*> &Robots, std::vector <Programm \*> &Programms);

void prepare\_field(Field &); // расставить на поле статичные

bool is\_task\_completed(Field &, vector <Robot \*> &Robots); // проверка на выполненность

string get\_text\_task(); // вернуть текст задания

void draw\_an\_example() {}; // иллюстрирование решения задания(для художника)

};

2.3 Иерархия классов показана рисунке 1.

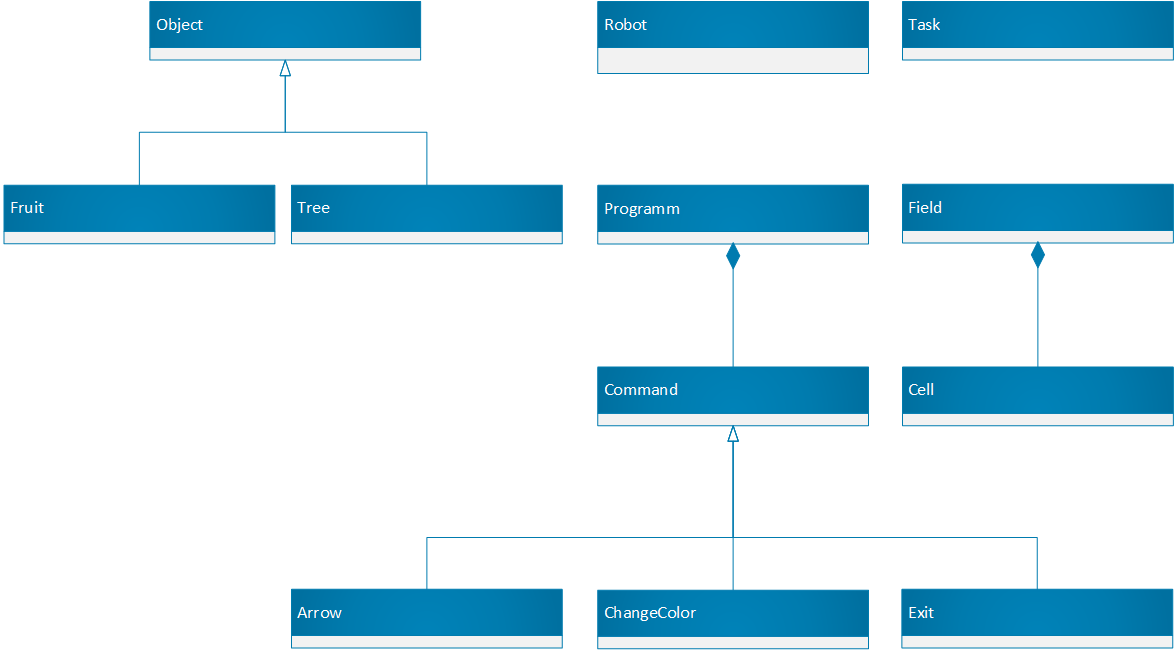


Рисунок 1 – Иерархия классов

**3** **ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ**

3.1 Требования к компьютеру:

Процессор: Intel Pentium 4 1ГГц или выше

Память: 512+ MБ

Видеокарта: разрешение 1024x768 или выше, 24-битный цвет

Свободное место на диске: 10 Мб

Операционная система: Windows XP или выше

Дополнительное устройство: клавиатура

3.2 Для установки скопировать следующие файлы на диск:

– prog.exe;

– r0.bmp, r1.bmp, r2.bmp, r3.bmp, wooden-crate\_blue.bmp, wooden-crate\_green.bmp, wooden-crate\_red.bmp, wooden-crate\_white.bmp, wooden-crate\_yellow.bmp, arrow\_blue.bmp, arrow\_green.bmp, arrow\_red.bmp, arrow\_white.bmp, arrow\_yellow.bmp.

**4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

При запуске программы откроется окно с игровым полем в левой части экрана, текстом задания в правой и индикатором выбранной программы для редактирования (см. рисунок 2).



Рисунок 2 – Окно редактора

Поле в левой части используется для решения задания. При нажатии клавиш строк изменяется выбираемая клетка поля, выделяемая жёлтым квадратом. При попытке выхода за границы поля сохраняется последняя корректная позиция выбранной клетки. Некоторые команды могут быть доступны для их перемещения и удаление, а также роботы для перемещения и смены направления движения, в зависимости от изначального задания.

Для выбора робота или команды нажмите на «Tab», после этого выбрать клетку, в которую данная команда будет перемещена при помощи стрелок.

Нажатием на клавишу «Delete» при выбранной одной из команд: стрелке, банка с краской или выход – удаляется соответствующая команда.

Для смены направления у робота нужно его выбрать, сменить направление клавишами «w», «a», «s», «d» для назначения направлений вверх, вправо, вниз или влево соответственно.

Нажатием на «Enter» запускается программы роботов.

Нажатием на «Backspace» поле возвращается в первоначальное состояние.

В случае столкновения 2-х роботов или робота с границей или препятствием, выведется сообщение об аварии и программа будет доступна для возвращения в первоначальное состояние.

Для завершения работы с программой необходимо щелкнуть по кнопке с крестиком в верхнем левом углу или при помощи клавиши «Esc».

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выявлены объекты предметной области и определена система классов для них, разработан интерфейс программы. После объектно-ориентированного проектирования классы были реализованы на языке С++. Разработанный код был проверен на контрольных тестах и в код были внесены необходимые исправления. Для программы была разработана документация, описывающая её установку и использование. Таким образом, цель работы была достигнута, задачи – решены.

Результаты работы можно использовать в процессе последующего обучения в форме навыков практического применения объектно-ориентированного подхода для разработки сложных программных систем, понимания порядка этапов разработки программного обеспечения и достигаемых на каждом этапе результатов.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Графическая библиотека WinBGIm. – URL: https://ipc.susu.ru/20786.html (дата обращения: 20.05.2022).

2 Демидов, А. К. Объектно-ориентированное программирование на C++.[Текст]:учеб. пособие. – URL: [ftd (susu.ru)](https://lib.susu.ru/ftd?base=SUSU_METHOD&key=000557669&dtype=F&etype=.pdf) (дата обращения: 20.05.2022).

3 Липман, С. Язык программирования С++. Полное руководство. [Электронный ресурс] / С. Липман, Ж. Лажойе. — Электрон. дан. — М. : ДМК Пресс, 2006. — 1105 с. – URL: <http://e.lanbook.com/book/1216> (дата обращения: 20.05.2022).

# ПРИЛОЖЕНИЕ А

А.1 Файл main.cpp

#include "Objects.hpp"

using namespace std;

vector <Robot \*> Robots; // глобальный вектор с роботами

vector <Programm \*> Programms; // глобальный вектор с программами

Field field(WIDTH\_I, HEIGHT\_J); //создание объекта поля

//очистка окна

void clearWin() {

putimage(0, 0, loadBMP("inteface.bmp"), COPY\_PUT);

}

//вставка текста задания

void put\_text(string text\_task) {

setbkcolor(NO\_COLOR);

setcolor(BLACK);

const char \*cstr = text\_task.c\_str();

settextstyle(GOTHIC\_FONT, HORIZ\_DIR, 12);

outtextxy(510, 60, cstr);

}

//отрисовка индикатора выбранной программы в нижней панели

void drawBarCurrectProg(int i) {

setcolor(BLACK);

int heightB = 125;

rectangle(i\*heightB,500, i\*heightB+122, 549);

setfillstyle(SOLID\_FILL, Programms[i]->get\_col());

bar(i\*heightB,501, i\*heightB+122, 549);

}

// перерисовка всех элементов во время выполнения программы роботами

void reDraw(string text\_task) {

clearWin(); // отрисовываем задний фон

// выводим текст задания

put\_text(text\_task);

//прорисовка объектов на поле

field.draw();

// Отрисовываем программмы и роботов

for (int i = 0; i < Programms.size(); i++) {

Programms[i]->draw();

drawBarCurrectProg(i);

}

for (int i = 0; i < Robots.size(); i++)

Robots[i]->draw();

}

// выделение ячейки

void highlightCell(position current\_cell) {

setlinestyle(0, 1, 3);

setcolor(YELLOW);

rectangle(100\*current\_cell.x, 100\*current\_cell.y, 100\*current\_cell.x+100, 100\*current\_cell.y+100);

}

//отрисока текущей программы

void drawCurrectProg(int n\_currect\_com, position current\_cell) {

clearWin();

Programms[n\_currect\_com]->draw();

field.draw();

for (int i = 0; i < Robots.size(); i++)

if (Robots[i]->get\_color() == Programms[n\_currect\_com]->get\_col()) {

Robots[i]->draw();

drawBarCurrectProg(n\_currect\_com);

}

}

int n\_currect\_com = 0; //номер текущей программы

int main() {

initwindow(800, 550, "Программа для обучения программированию");

clearWin();

position current\_cell(0, 0); // позиция текущей клетки

Task task("tast\_list.txt");

task.initialize(field, Robots, Programms); //инициализация роботов и программ

string text\_task = task.get\_text\_task(); // получение текста задания

task.prepare\_field(field); // подготовка поля с расстановкой объектов

highlightCell(current\_cell); // отображение текущей ячейки

while (1) {

//выбор текущей клетки

switch (getch(kbhit())) {

case KEY\_UP:

drawCurrectProg(n\_currect\_com, current\_cell);

current\_cell.y += (current\_cell.y == 0) ? 0 : -1;

highlightCell(current\_cell);

break;

case KEY\_DOWN:

drawCurrectProg(n\_currect\_com, current\_cell);

current\_cell.y += (current\_cell.y == HEIGHT\_J-1) ? 0 : 1;

highlightCell(current\_cell);

break;

case KEY\_LEFT:

drawCurrectProg(n\_currect\_com, current\_cell);

current\_cell.x += (current\_cell.x == 0) ? 0 : -1;

highlightCell(current\_cell);

break;

case KEY\_RIGHT:

drawCurrectProg(n\_currect\_com, current\_cell);

current\_cell.x += (current\_cell.x == WIDTH\_I-1) ? 0 : 1;

highlightCell(current\_cell);

break;

// отображение программ только определённого цвета

case '1':

cout << "Выбрана программа 1"<<endl;

n\_currect\_com = 0;

drawCurrectProg(n\_currect\_com, current\_cell);

put\_text(text\_task);

break;

case '2':

if (Programms.size() <2)

break;

cout << "Выбрана программа 2"<<endl;

n\_currect\_com = 1;

drawCurrectProg(n\_currect\_com, current\_cell);

put\_text(text\_task);

break;

case '3':

if (Programms.size() <3)

break;

cout << "Выбрана программа 3"<<endl;

n\_currect\_com = 2;

drawCurrectProg(n\_currect\_com, current\_cell);

put\_text(text\_task);

break;

case '4':

if (Programms.size() <4)

break;

cout << "Выбрана программа 4"<<endl;

n\_currect\_com = 3;

drawCurrectProg(n\_currect\_com, current\_cell);

put\_text(text\_task);

break;

//выбор комманды

case KEY\_ENTER:

break;

// удаление объкта

case KEY\_DELETE:

Programms[n\_currect\_com]->delete\_com(Programms[n\_currect\_com]->select(current\_cell));

drawCurrectProg(n\_currect\_com, current\_cell);

break;

// запуск программы

case KEY\_TAB:

// перемещение роботов

for (int i = 0; i < Robots.size(); i++) {

position new\_pos = Robots[i]->get\_cordinat();

new\_pos.x += Robots[i]->get\_direction().x;

new\_pos.y += Robots[i]->get\_direction().y;

Robots[i]->set\_cordinat(new\_pos);

Robots[i]->draw();

reDraw(text\_task);

}

break;

//отмена удаления

case KEY\_BACKSPACE:

break;

//закрытие

case KEY\_ESC:

closegraph();

return 0;

}

}

}

А.2 Файл Objects.hpp

#ifndef OBJECTS\_H

#define OBJECTS\_H

#include <fstream>

#include <iostream>

#include <format>

#include <algorithm>

#include <string>

#include <typeinfo>

#include <time.h>

#include <stdlib.h>

#include "graphics.h"

#define WIDTH\_I 5

#define HEIGHT\_J 5

#define COUNFUITS 2

#define COUNTTREE 1

using namespace std;

class Robot;

class Robots;

class Object;

struct Cell;

class Field;

class Сommand;

class Programm;

class Task;

//струкутура для хранения координат парами

struct position {

int x, y; // координаты

position() {x = y = 0;} // конструктор без передаваемых парамтров

position(int new\_x, int new\_y):x(new\_x), y(new\_y) {}

position(const position &pos) {x = pos.x;y = pos.y;} // конструктор копий

// перегрузка оператора для сравнения

friend auto operator<=>(const position&, const position&) = default;

};

class Robot {

IMAGE \*img; // картинка робота

position pos; //текущие координаты робота;

position direction; // текущее направление(смещение по координатам) {0, 1}/{1, 0}/{0, -1}/{-1, 0}

int color; // текущий цвет

bool allow\_change\_direction; // разрешено изменять направление?

bool allow\_change\_cordinat; // разрешено изменять координаты?

public:

Robot(IMAGE \*img\_robot,bool is\_allow\_change\_direction, bool is\_allow\_change\_cordinat); //конструктор

~Robot(); //деструктор

void set\_cordinat(position); // установить координаты

void set\_direction(position); // установить направление

void set\_color(int); // установить цвет

int get\_color(); // вернуть цвет

position get\_cordinat(); // вернуть координаты

position get\_direction(); // вернуть направление

void change\_Field(Field &); // перед выходом из клетки удаление или замена объекта

void draw();

private:

bool is\_crash(vector <Robot \*> &Robots); // столкнулся(набор роботов)?

};

//базовый класс для неподвижных сущностей

class Object {

protected:

IMAGE \*img; // картинка объекта

public:

Object(IMAGE \*); // конструктор

Object(const Object &obj); // Конструктор копий

~Object();// деструтор

virtual void draw(position) = 0; // вернуть картин

virtual bool is\_access(Robot &) = 0; // проверка клетки на доступность для робота

};

// объект доступный для сбора

class Fruit : public Object {

public:

Fruit(IMAGE \*);

~Fruit();

void draw(position);

bool is\_access(Robot &);

};

// объект не доступный для перемещения

class Tree : public Object {

public:

Tree(IMAGE \*);

~Tree();

void draw(position);

bool is\_access(Robot &);

};

// струкура для клетки поля

struct Cell {

Object \*current\_object = nullptr; // объект в клетке

int color = WHITE; // цвет клетки

};

//класс поля

class Field {

int width, height; // размеры поля

vector<vector<Cell>> fullField; //поле из клеток

public:

Field(int, int); // констуктор

void set\_obj(Object \*, position); // установить объект

void delete\_obj(position); // удалить объект

void set\_color(position, int); // установить цвет

void draw(); // отрисовка всех Object на поле

Object \*get\_object(position); // получить объкт на клетке

};

class Command {

bool is\_allow\_change\_cordinat; //разрешено изменять координаты?

bool is\_allow\_delete; //разрешено удалять?

friend class Programm;

protected:

IMAGE \*img; // изображение комманды

position coord; //координаты комманды

public:

Command(bool, bool, position); //is\_allow\_change\_cordinat, is\_allow\_delete, x, y

Command(const Command &com); // Конструктор копий

Command() = default; // Конструктор копий

void set\_pos(position); // задать новую позицию комманде

position get\_pos(); // вернуть текущую позицию

virtual void use(Robot &) = 0; // виртуальный метод на воздействие на робота

virtual void draw(int color) = 0; // виртуальный метод рисования

};

// "Стрелка", меняющая нарпавление

class Arrow : public Command {

position orientation;

public:

// разрешение на пермещение и удаление, позиция расположения, позимещения изменения направления

Arrow(bool is\_allow\_change\_cordinat, bool is\_allow\_delete, position coord, position orient);

void use(Robot &);

void draw(int color) ;

};

//"Банка с краской", меняющая цвет робота

class ChangeColor : public Command {

int color;

public:

ChangeColor(bool is\_allow\_change\_cordinat, bool is\_allow\_delete, position, int color);

void use(Robot &);

void draw(int color);

};

//"Выход", удаляющий робота

class Exit : public Command {

public:

Exit(bool is\_allow\_change\_cordinat, bool is\_allow\_delete, position);

void use(Robot &);

void draw(int color);

};

// Класс программ, содержащий набор комманд одного цвета

class Programm {

int color = WHITE; // цвет программы

vector<Command \*> commands ; // вектор комманд внутри одной программы

public:

Programm(int color); // конструктор

int get\_col(); // вернуть цвет

void add(Command \*);// добавление комманды

void draw(); // отрисовать все комманды данной программы

Command \*select(position); // выбрать команду в position

void delete\_com(Command\*); // удаление комманды

};

extern vector <Robot\*> Robots; // глобальный вектор с роботами

extern vector <Programm\*> Programms; // глобальный вектор с программами

class Task {

string text\_task; // текст задания

string name\_taskFile; // название файла с заданием

int count\_robots; // количество роботов

int count\_commands;// количество комманд

int count\_tree; // количество деревьев

int count\_fruit; // количество урожая

// инициализация всех компанентов согласно заданию

public:

Task(const string);// название файла с заданием

// инициализация робтов и программ по файлу с заданием

void initialize(Field &, std::vector <Robot \*> &Robots, std::vector <Programm \*> &Programms);

void prepare\_field(Field &); // расставить на поле статичные

bool is\_task\_completed(Field &, vector <Robot \*> &Robots); // проверка на выполненность

string get\_text\_task(); // вернуть текст задания

void draw\_an\_example() {}; // иллюстрирование решения задания(для художника)

};

#endif

А.3 Файл Objects.cpp

#include "Objects.hpp"

constexpr int color\_prog[4] = {RED, GREEN, BLUE, YELLOW};

vector<position> places\_taken;

using namespace std;

int direct2grad(position direct) {

if (direct.x == -1)

return 270;

if (direct.y == 1)

return 180;

if (direct.y == -1)

return 90;

else

return 0;

}

Robot::Robot(IMAGE \*image, bool direction, bool cordinat):img(image), allow\_change\_direction(direction), allow\_change\_cordinat(cordinat) {}

Robot::~Robot() {freeimage(img);}

void Robot::set\_cordinat(position new\_pos) {

pos = position(new\_pos.x, new\_pos.y);

}

void Robot::set\_direction(position new\_direction) {

direction = position(new\_direction.x, new\_direction.y);

}

void Robot::set\_color(int new\_color) {color = new\_color;}

int Robot::get\_color() {return color;}

position Robot::get\_cordinat() {return pos;}

position Robot::get\_direction() {return direction;}

void Robot::change\_Field(Field &Field) {

}

void Robot::draw() {

IMAGE \*rotImg = imageturn(img, direct2grad(pos), WHITE);

putimage(pos.x\*100+5,pos.y\*100+5, rotImg, COPY\_PUT);

}

bool Robot::is\_crash(std::vector <Robot \*> &Robots) {

//if (

return 1;

}

Object::Object(IMAGE \*image): img(image) {}

Object::Object(const Object &obj) {img = obj.img;}

Object::~Object() {freeimage(img);}

Fruit::Fruit(IMAGE \*image): Object(image) {}

Fruit::~Fruit() {freeimage(img);}

Tree::Tree(IMAGE \*image): Object(image) {}

Tree::~Tree() {freeimage(img);}

bool Fruit::is\_access(Robot &current\_robot) {return 1;}

void Fruit::draw(position pos) {return putimage(pos.x, pos.y, img, COPY\_PUT);}

bool Tree::is\_access(Robot &current\_robot) {return 0;}

void Tree::draw(position pos) { putimage(pos.x, pos.y, img, COPY\_PUT);}

Field::Field(int w, int h):width(w), height(h), fullField(width, vector<Cell>(height)) {}

void Field::set\_obj(Object \*obj, position pos) { // i, j

fullField[pos.x][pos.y].current\_object = obj;

}

void Field::delete\_obj(position pos) { // i, j

fullField[pos.x][pos.y].current\_object = nullptr;

}

void Field::set\_color(position pos, int color) { // i, j

fullField[pos.x][pos.y].color = color;

}

Object \*Field::get\_object(position pos) {

return fullField[pos.x][pos.y].current\_object;

}

void Field::draw() {

for (int i = 0; i < width; i++) {

for (int j = 0; j < height; j++) {

if (fullField[i][j].current\_object != nullptr)

fullField[i][j].current\_object->draw(position(i\*100+5, j\*100+5));

}

}

}

Command::Command(bool is\_change\_cordinat, bool is\_delete, position new\_coord):

is\_allow\_change\_cordinat(is\_change\_cordinat), is\_allow\_delete(is\_delete), coord(new\_coord) {}

Command::Command(const Command &com) {

is\_allow\_change\_cordinat = com.is\_allow\_change\_cordinat;

is\_allow\_delete = com.is\_allow\_delete;

coord = com.coord;

}

void Command::set\_pos(position pos) {

coord.x = pos.x;

coord.y = pos.y;

}

position Command::get\_pos() {return coord;}

Arrow::Arrow(bool is\_change\_cordinat, bool is\_delete, position new\_coord, position direction):

Command(is\_change\_cordinat, is\_delete, new\_coord), orientation(direction) {}

ChangeColor::ChangeColor(bool is\_change\_cordinat, bool is\_delete, position new\_coord, int color):

Command(is\_change\_cordinat, is\_delete, new\_coord), color(color) {}

Exit::Exit(bool is\_change\_cordinat, bool is\_delete, position new\_coord):

Command(is\_change\_cordinat, is\_delete, new\_coord) {}

void Arrow::use(Robot &robot) {

robot.set\_direction(this->orientation);

}

void Arrow::draw(int col) {

switch (col) {

case RED:

this->img = loadBMP("arrow\_red.bmp");

break;

case BLUE:

this->img = loadBMP("arrow\_blue.bmp");

break;

case GREEN:

this->img = loadBMP("arrow\_green.bmp");

break;

case YELLOW:

this->img = loadBMP("arrow\_yellow.bmp");

break;

default:

this->img = loadBMP("arrow\_white.bmp");

}

IMAGE \*rotImg = imageturn(this->img, direct2grad(orientation), WHITE);

putimage(coord.x\*100+10,coord.y\*100+10, rotImg, COPY\_PUT);

}

void ChangeColor::use(Robot &robot) {

robot.set\_color(this->color);

}

void ChangeColor::draw(int col) {

//cout <<"coord = ("<< coord.x <<", "<< coord.y<<")\n";

setcolor(BLACK);

setfillstyle(SOLID\_FILL, col);

bar(coord.x\*100+25, coord.y\*100+10, coord.x\*100+75, coord.y\*100+90);

setfillstyle(SOLID\_FILL, color);

fillellipse(coord.x\*100+50, coord.y\*100+50, 20, 20);

}

void Exit::use(Robot &robot) {

Robots.erase(ranges::find(Robots, &robot));

}

void Exit::draw(int col) {

switch (col) {

case RED:

this->img = loadBMP("wooden-crate\_red.bmp");

break;

case BLUE:

this->img = loadBMP("wooden-crate\_blue.bmp");

break;

case GREEN:

this->img = loadBMP("wooden-crate\_green.bmp");

break;

case YELLOW:

this->img = loadBMP("wooden-crate\_yellow.bmp");

break;

default:

this->img = loadBMP("wooden-crate\_white.bmp");

}

putimage(coord.x\*100+5, coord.x\*100+5, img, COPY\_PUT);

}

Programm::Programm(int c): color(c) {}

int Programm::get\_col() {return color;}

void Programm::add(Command \*command) {

commands.push\_back(command);

}

void Programm::draw() {

for (int i = 0; i < commands.size(); i ++) {

cout<< "color = " << color << ", ";

commands[i]->draw(color);

}

}

void Programm::delete\_com(Command \*com) {

commands.erase(ranges::find(commands, com));

}

Command \*Programm::select(position pos) {

auto it = find\_if(commands.begin(), commands.end(), [pos](Command \*com) -> bool {

return com->get\_pos() == pos;});

if (it != commands.end())

cout << "command is find\n";

else

cout << "command is't find\n";

}

Task::Task(const string file\_name): name\_taskFile(file\_name) {}

void Task::initialize(Field &field, vector <Robot \*> &Robots,vector <Programm \*> &Programms) {

ifstream file;

setlocale(LC\_ALL, "Russian");

file.open(name\_taskFile);

getline(file, text\_task);

file >> count\_robots >> count\_commands;

cout << text\_task << endl;

cout << count\_robots<< " " << count\_commands<< endl;

for (int i = 0; i < count\_robots; i++) {

int r\_x, r\_y, r\_color;

int f\_color, f\_direct;

string f\_change\_direct, f\_change\_coord;

file >> r\_x >> r\_y;

file >>f\_color >> f\_direct;

file >> f\_change\_direct >> f\_change\_coord;

cout << r\_x << " " << r\_y << " " << f\_color << " "<< f\_direct <<" "<< f\_change\_direct <<" "<< f\_change\_coord << endl;

bool change\_direct, change\_coord;

position direct;

change\_direct = (f\_change\_direct == "да"? true : false);

change\_coord = (f\_change\_coord == "да"? true : false);

switch (f\_direct) {

case 0:

direct = position(1, 0);

break; // "вправ"

case 1:

direct = position(0, 1);

break; //"вверх"

case 2:

direct = position(-1, 0);

break; //"влево"

case 3:

direct = position(0, -1);

break; //"вниз"

}

char name\_image[7];

snprintf(name\_image, sizeof(name\_image), "r%d.bmp",f\_color);

Robot \*new\_robot = new Robot(loadBMP(name\_image),change\_direct, change\_coord);

new\_robot->set\_color(color\_prog[f\_color]);

new\_robot->set\_cordinat(position(r\_x, r\_y));

new\_robot->set\_direction(direct);

Robots.push\_back(new\_robot);

}

// цикл чтения информации о программах

for (int i = 0; i < count\_commands; i++) {

string name\_com;

int com\_x, com\_y;

int f\_color;

file >> name\_com;

string f\_allow\_delete, f\_change\_coord;

file >>f\_color >> com\_x >> com\_y;

file >> f\_change\_coord >> f\_allow\_delete;

bool allow\_delete = (f\_allow\_delete == "да"? true : false);

bool change\_coord = (f\_change\_coord == "да"? true : false);

cout << name\_com << " " << f\_color <<" "<< com\_x << " " << com\_y << " "

<< f\_change\_coord <<" "<< f\_allow\_delete << " ";

Command \*command;

places\_taken.push\_back(position(com\_x, com\_y));

if (name\_com == "стрелка") {

int f\_orient;

position orient;

file >> f\_orient;

cout << f\_orient << endl;

switch (f\_orient) {

case 0:

orient = position(1, 0);

break; // "вправo"

case 1:

orient = position(0, 1);

break; //"вверх"

case 2:

orient = position(-1, 0);

break; //"влево"

case 3:

orient = position(0, -1);

break; //"вниз"

}

Arrow \*arrow = new Arrow(change\_coord, allow\_delete, position(com\_x, com\_y), orient);

command = arrow;

}

else if (name\_com == "банка\_с\_краской") {

int f\_change\_col;

file >> f\_change\_col;

cout << f\_change\_col << endl;

ChangeColor \*canOfPaint = new ChangeColor(change\_coord, allow\_delete, position(com\_x, com\_y), color\_prog[f\_change\_col]);

command = canOfPaint;

}

else if (name\_com == "выход") {

cout << endl;

Exit \*box = new Exit(change\_coord, allow\_delete, position(com\_x, com\_y));

command = box;

}

else

cout << "Сюда вставить проверку на ошибки\n";

auto it = find\_if(Programms.begin(), Programms.end(), [f\_color](Programm \*prog) -> bool {

return prog->get\_col() == color\_prog[f\_color];});

if (it != Programms.end())

{

cout << "Нашёл" << endl;

(\*it)->add(command);

}

else {

cout << "Не нашёл" << endl;

Programm \*new\_programm = new Programm(color\_prog[f\_color]);

new\_programm->add(command);

Programms.push\_back(new\_programm);

}

}

cout << "Чтение файла завешенно. Закрываем файл\n" << endl;

file.close();

}

void Task::prepare\_field(Field &field) {

count\_fruit = COUNFUITS;

count\_tree = COUNTTREE;

srand(time(NULL));

for (int n\_fruit = 0; n\_fruit < count\_fruit;)

{

position pos\_fruit;

pos\_fruit.x = rand() % (WIDTH\_I);

pos\_fruit.y = rand() % (HEIGHT\_J);

auto it = find\_if(places\_taken.begin(), places\_taken.end(),

[pos\_fruit](position place) -> bool {return (place.x != pos\_fruit.x&& place.y != pos\_fruit.y);});

if (it != places\_taken.end()) {

n\_fruit ++;

places\_taken.push\_back(pos\_fruit);

Fruit \*fruit =new Fruit(loadBMP("apple.bmp"));

field.set\_obj(fruit, pos\_fruit);

}

}

for (int n\_fruit = 0; n\_fruit < count\_tree;)

{

position pos\_tree;

pos\_tree.x = rand() % (WIDTH\_I-1);

pos\_tree.y = rand() % (HEIGHT\_J-1);

auto it = find\_if(places\_taken.begin(), places\_taken.end(),

[pos\_tree](position place) -> bool {return (place.x != pos\_tree.x&& place.y != pos\_tree.y);});

if (it != places\_taken.end()) {

n\_fruit ++;

places\_taken.push\_back(pos\_tree);

field.set\_obj(new Tree(loadBMP("tree.bmp")), pos\_tree);

}

}

}

string Task::get\_text\_task() {

return text\_task;

}