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 программа для обучения основам программирования, код которой приводится в приложении.

ОГЛАВЛЕНИЕ

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 до-
ступность 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 воздей-
ствие 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, posi-
tion 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, posi-
tion); // 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 про-
граммы
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 програм-
мами

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.

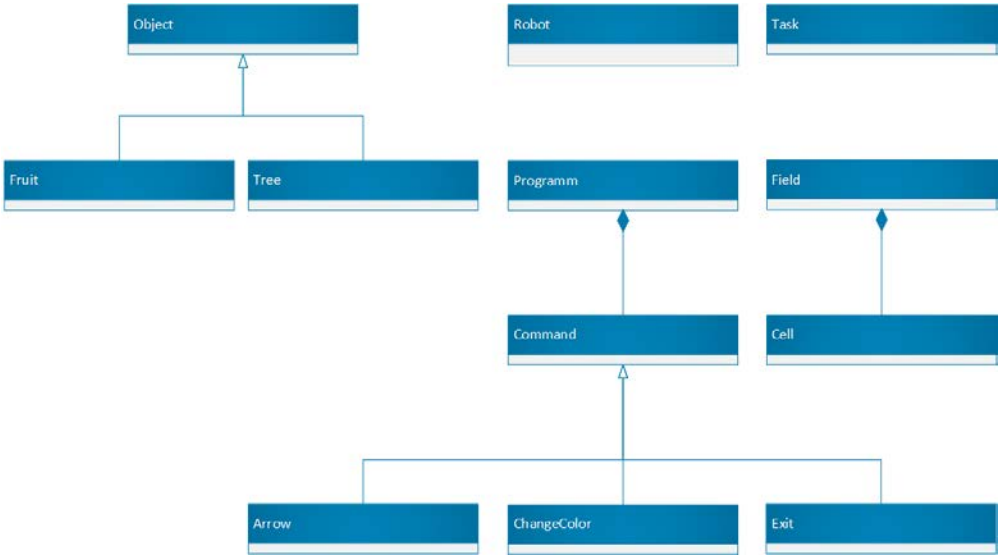

Рисунок 1 – Иерархия классов

3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ

3.1 Требования к компьютеру:
Процессор: Intel Pentium 4 1ГГц или выше
Память: 512+ МБ
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 программы

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 C++. 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) (дата обращения: 20.05.2022).
3 Липман, С. Язык программирования C++. 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;
         }
      }
}
```

A.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 Command;
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
```

A.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; // "вправо"
         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;
}
```