Результат разработки иерархии и интерфейса классов:

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) //от рисовка 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) //отрисока текущей программы
int main()
```

2.Модуль Objects (интерфейсная часть в файле Objects.hpp, реализация в Objects.cpp) содержит следующие классы, функции и константы:

```
//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)
};
```

Иерархия классов