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) //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()

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

//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)

};

Иерархия классов

