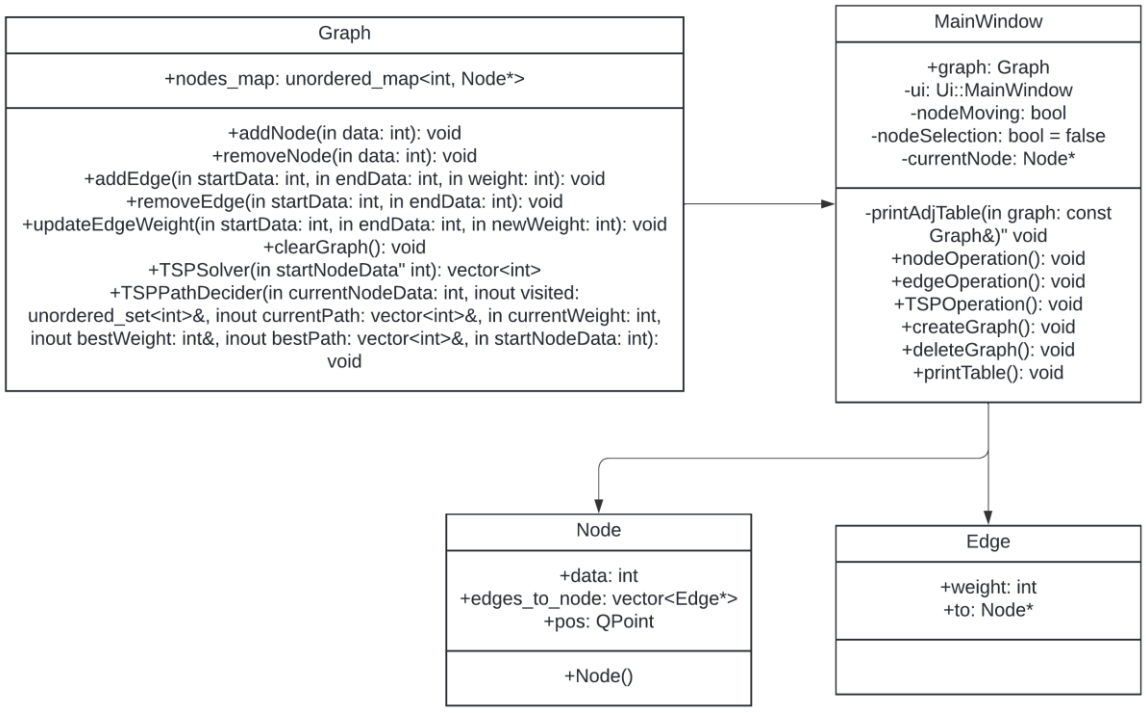
Романов Артём Алексеевич (ИВТ-23-1Б). АРМ менеджера товарного склада + задача коммивояжёра (вариант 14).

Ссылка на видео с демонстрацией работы программ:

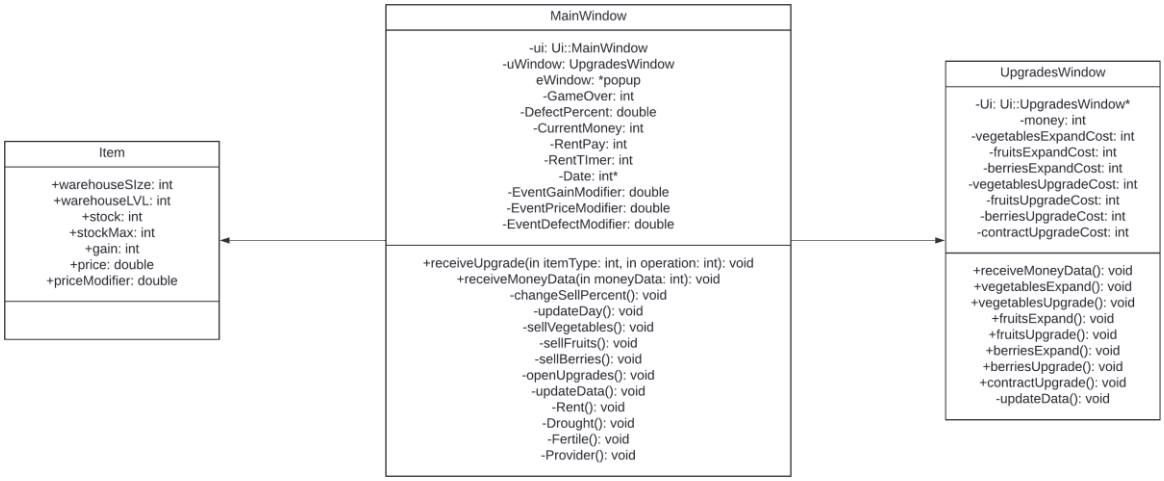
https://youtu.be/zT6jT3AkHOI

UML-Диаграммы:

Коммивояжёр:



АРМ менеджера склада:



Код коммивояжёра:

Mainwindow.h

```cpp
#include <QWidget>
#include <QMouseEvent>
#include <ui_mainwindow.h>
#include <unordered_map>
#include <unordered_set>

using namespace std;

// Предопределение классов
class Edge;
class Node;
class Graph;

class Node{
public:
    int data;
    vector<Edge*> edges_to_node;
    QPoint pos;
    Node(){
        pos = QPoint(800 + (rand()%400 - 400), 300 + (rand()%300 - 300));
    }
};

class Edge{
public:
    int weight;
    Node* to;
};

class Graph{
public:
    unordered_map<int, Node*> nodes_map;

    void addNode(int data);
    void removeNode(int data);

    void addEdge(int fromData, int toData, int weight);
    void removeEdge(int startData, int endData);
    void updateEdgeWeight(int startData, int endData, int newWeight);

    void clearGraph();

    vector<int> TSPSolver(int startNodeData);
    void TSPPathDecider(int currentNodeData, unordered_set<int>& visited, vector<int>& currentPath, int currentWeight, int& bestWeight, vector<int>& bestPath, int startNodeData);
};

class MainWindow : public QMainWindow{
    Q_OBJECT
public:
    MainWindow(QWidget* parent = nullptr);
    ~MainWindow();
    Graph graph;

protected:
    void paintEvent(QPaintEvent* event) override;
    void mousePressEvent(QMouseEvent* event) override;
    void mouseMoveEvent(QMouseEvent* event) override;
    void mouseReleaseEvent(QMouseEvent* event) override;
```

```cpp
private:
    Ui::MainWindow ui;
    Node* selectedNode;
    bool nodeMoving;
    bool nodeSelection = false;
    Node* sNode;

    void printAdjTable(const Graph& graph);

public slots:
    void nodeOperation();
    void edgeOperation();
    void TSPOperation();
    void createGraph();
    void deleteGraph();
    void printTable();
};
```

Main.cpp:

```cpp
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Mainwindow.cpp:

```cpp
#include "mainwindow.h"

#include <QPainter>
#include <cmath>
#include <QTimer>
#include <queue>
#include <stack>

MainWindow::MainWindow(QWidget* parent): QMainWindow(parent){
    ui.setupUi( this );

    connect(ui.nodeOperationButton, &QPushButton::pressed, this, &MainWindow::nodeOperation);
    connect(ui.edgeOperationButton, &QPushButton::pressed, this, &MainWindow::edgeOperation);
    connect(ui.TSPButton, &QPushButton::pressed, this, &MainWindow::TSPOperation);
    connect(ui.createGraphButton, &QPushButton::clicked, this, &MainWindow::createGraph);
    connect(ui.deleteGraphButton, &QPushButton::clicked, this, &MainWindow::deleteGraph);
    connect(ui.printTableButton, &QPushButton::clicked, this, &MainWindow::printTable);
}

// Операции над узлами
void Graph::addNode(int data){
    if (nodes_map.find(data) == nodes_map.end()){
        Node* newNode = new Node;
        newNode->data = data;
        nodes_map[data] = newNode;
    }
}
void Graph::removeNode(int data){
    for (auto& pair : nodes_map){
        Node* node = pair.second;
        vector<Edge*> edges_to_remove;
        for (Edge* edge : node->edges_to_node){
            if (edge->to->data == data){
                edges_to_remove.push_back(edge);
            }
        }
        for (Edge* edge : edges_to_remove){
            auto it = find(node->edges_to_node.begin(), node->edges_to_node.end(), edge);
            if (it != node->edges_to_node.end()){
                node->edges_to_node.erase(it);
                delete edge;
            }
        }
    }
    auto it = nodes_map.find(data);
    if (it != nodes_map.end()){
        delete it->second;
        nodes_map.erase(it);
    }
}
void MainWindow::nodeOperation(){
    if (ui.nodeValue->text().isEmpty()){ return; }

    int operation = ui.nodeOperations->currentIndex();
    int nodeValue = ui.nodeValue->text().toInt();

    // 0 - Добавить, 1 - Удалить
    switch(operation){
    case 0: graph.addNode(nodeValue); break;
    case 1: graph.removeNode(nodeValue); break;
    }
    ui.nodeValue->clear(); update();
    ui.statusText->setText("Операция над узлом проведена.");
}
```

```cpp
// Операции над рёбрами
void Graph::addEdge(int fromData, int toData, int weight){
    for (Edge* edge : nodes_map[fromData]->edges_to_node){
        if (edge->to == nodes_map[toData]){
            return;
        }
    }
    Edge* newEdge = new Edge();
    newEdge->to = nodes_map[toData];
    newEdge->weight = weight;
    nodes_map[fromData]->edges_to_node.push_back(newEdge);
}
void Graph::removeEdge(int startData, int endData){
    auto startNodeIt = nodes_map.find(startData);
    auto endNodeIt = nodes_map.find(endData);
    if (startNodeIt == nodes_map.end() || endNodeIt == nodes_map.end())
    {
        return;
    }
    Node* startNode = startNodeIt->second;
    Edge* edgeToRemove = nullptr;

    for (Edge* edge : startNode->edges_to_node)
    {
        if (edge->to->data == endData)
        {
            edgeToRemove = edge;
            break;
        }
    }
    if (edgeToRemove)
    {
        auto it = find(startNode->edges_to_node.begin(), startNode->edges_to_node.end(), edgeToRemove);
        if (it != startNode->edges_to_node.end())
        {
            startNode->edges_to_node.erase(it);
            delete edgeToRemove;
        }
    }
}
void Graph::updateEdgeWeight(int startData, int endData, int newWeight){
    if (nodes_map.find(startData) == nodes_map.end() || nodes_map.find(endData) == nodes_map.end()){
        return;
    }
    Node* startNode = nodes_map[startData];
    Node* endNode = nodes_map[endData];
    for (Edge* edge : startNode->edges_to_node){
        if (edge->to == endNode){
            edge->weight = newWeight;
            return;
        }
    }
}
```

```
void MainWindow::edgeOperation(){
    if (ui.nodeStart->text().isEmpty() || ui.nodeEnd->text().isEmpty()) { return; }

    int nodeStart = ui.nodeStart->text().toInt();
    int nodeEnd = ui.nodeEnd->text().toInt();
    int edgeWeight = ui.edgeWeight->text().toInt();

    int operation = ui.edgeOperations->currentIndex();

    // 0 - Добавить, 1 - Удалить, 2 - Редактировать вес
    switch(operation){
    case 0: if(ui.edgeWeight->text().isEmpty()){ return; } graph.addEdge(nodeStart, nodeEnd, edgeWeight);  break;
    case 1: graph.removeEdge(nodeStart, nodeEnd); break;
    case 2: if(ui.edgeWeight->text().isEmpty()){ return; } graph.updateEdgeWeight(nodeStart, nodeEnd, edgeWeight);  break;

    ui.nodeStart->clear(); ui.nodeEnd->clear(); ui.edgeWeight->clear(); update();
    ui.statusText->setText("Операция над гранью проведена.");
    }
}

// Удаление и создание графа
void Graph::clearGraph(){
    for (auto& pair : nodes_map){
        Node* node = pair.second;
        delete node;
    }

    nodes_map.clear();
}
void MainWindow::createGraph(){
    graph.addNode(1);
    graph.addNode(2);
    graph.addNode(3);
    graph.addNode(4);
    graph.addNode(5);
    graph.addNode(6);

    graph.addEdge(1, 2, 2);
    graph.addEdge(1, 6, 57);

    graph.addEdge(2, 1, 2);
    graph.addEdge(2, 6, 13);
    graph.addEdge(2, 4, 8);
    graph.addEdge(2, 3, 3);

    graph.addEdge(3, 2, 3);
    graph.addEdge(3, 4, 5);

    graph.addEdge(4, 3, 5);
    graph.addEdge(4, 2, 8);
    graph.addEdge(4, 6, 21);
    graph.addEdge(4, 5, 34);

    graph.addEdge(5, 6, 45);
    graph.addEdge(5, 4, 34);

    graph.addEdge(6, 1, 57);
    graph.addEdge(6, 2, 13);
    graph.addEdge(6, 4, 21);
    graph.addEdge(6, 5, 45);


    update();
    ui.statusText->setText("Граф задания создан.");
}
```

```cpp
void MainWindow::deleteGraph(){
    graph.clearGraph();

    ui.statusText->setText("Граф удалён.");
    update();
}

// Визуализация графа
void MainWindow::paintEvent(QPaintEvent* event){
    QPainter painter(this);

    QFont font = painter.font();
    font.setPointSize(16);
    painter.setFont(font);
    painter.setPen(QPen(Qt::cyan));

    for (const auto& pair : graph.nodes_map) {
        Node* node = pair.second;

        for (Edge* edge : node->edges_to_node) {
            painter.setOpacity(0.2);
            QPoint edgeStart;
            QPoint edgeEnd;

            double angles = atan2(-(edge->to->pos.y() - node->pos.y()), (edge->to->pos.x()-node->pos.x()));

            edgeStart = QPoint(node->pos.x()+20*cos(angles), node->pos.y() - 20*sin(angles));
            edgeEnd = QPoint(edge->to->pos.x()- 20 * cos(angles), edge->to->pos.y() + 20*sin(angles));

            painter.drawLine(edgeStart, edgeEnd);

            int x_t = edgeStart.x() + 4 * (edgeEnd.x() - edgeStart.x()) / 5 ;
            int y_t = edgeStart.y() - 4 * (edgeStart.y() - edgeEnd.y()) / 5;

            painter.setPen(QPen(Qt::black, 2));
            painter.setOpacity(1);
            painter.drawText(x_t-10, y_t+10, QString::number(edge->weight));
            painter.setOpacity(0.2);
            painter.setPen(QPen(Qt::cyan, 2));

            QLine line(edgeStart, edgeEnd);

            double angle = atan2(-line.dy(), line.dx())-M_PI/2;
            double arrowSize = 20;

            QPointF arrowP1 = edgeEnd + QPointF(sin(angle - M_PI / 12) * arrowSize, cos(angle - M_PI / 12) * arrowSize);
            QPointF arrowP2 = edgeEnd + QPointF(sin(angle + M_PI / 12) * arrowSize, cos(angle + M_PI / 12) * arrowSize);

            QPolygonF arrowHead;

            arrowHead << edgeEnd << arrowP1 << arrowP2;

            QPainterPath path;

            path.moveTo(edgeEnd);
            path.lineTo(arrowP1);
            path.lineTo(arrowP2);
            painter.fillPath(path, Qt::darkCyan);
            painter.drawPolygon(arrowHead);
            painter.setOpacity(1);
        }
    }
```

```
    painter.setBrush(Qt::NoBrush);
    painter.setPen(QPen(Qt::white, 2));

    for (const auto& pair : graph.nodes_map) {
        Node* node = pair.second;
        painter.drawEllipse(node->pos, 20, 20);
        painter.setPen(QPen(Qt::black, 2));
        painter.drawText(node->pos.x() - 9, node->pos.y() + 8, QString::number(node->data));
        painter.setPen(QPen(Qt::white, 2));
    }

    if (nodeSelection){
        painter.drawEllipse(100,100, 40, 40);
        painter.setBrush(Qt::yellow);
        painter.drawEllipse(sNode->pos, 20, 20);
        painter.setPen(QPen(Qt::black, 2));
        painter.drawText(sNode->pos.x() - 9, sNode->pos.y() + 8, QString::number(sNode->data));
    }
}
void MainWindow::mousePressEvent(QMouseEvent* event){
    if (event->button() == Qt::LeftButton){
        nodeMoving = false;
        for (const auto& pair : graph.nodes_map){
            Node* node = pair.second;
            if ((event->pos() - node->pos).manhattanLength() < 30){
                selectedNode = node;
                nodeMoving = true;
                break;
            }
        }
        update();
    }
}
void MainWindow::mouseMoveEvent(QMouseEvent* event){
    if (nodeMoving && selectedNode){
        selectedNode->pos = event->pos();
        update();
    }
}
void MainWindow::mouseReleaseEvent(QMouseEvent* event){
    if (event->button() == Qt::LeftButton && nodeMoving){
        nodeMoving = false;
        selectedNode = nullptr;
        update();
    }
}

// Вывод таблицы смежностей узлов
void MainWindow::printTable(){
    printAdjTable(graph);
}
```

```
void MainWindow::printAdjTable(const Graph& graph){
    QString result;
    for (const auto& pair : graph.nodes_map){
        int node = pair.first;
        Node* nodeConnections = pair.second;

        result += "Узел " + QString::number(node) + ": \n";
        unordered_set<int> printedNodes;
        for (Edge* edge : nodeConnections->edges_to_node){
            if (printedNodes.find(edge->to->data) == printedNodes.end()){
                result += "    Связь с " + QString::number(edge->to->data) + ", вес ребра - " + QString::number(edge->weight) + ". \n";
                printedNodes.insert(edge->to->data);
            }
        }
        ui.statusText->setText(result);
    }
}

// Задача коммивояжёра
vector<int> Graph::TSPSolver(int startNodeData){
    int bestWeight = numeric_limits<int>::max();

    vector<int> bestPath;
    unordered_set<int> visited;
    vector<int> currentPath;

    visited.insert(startNodeData);
    currentPath.push_back(startNodeData);

    TSPPathDecider(startNodeData, visited, currentPath, 0, bestWeight, bestPath, startNodeData);

    if (!bestPath.empty()) {
        bestPath.push_back(startNodeData);
    }

    return bestPath;
}
void Graph::TSPPathDecider(int currentNodeValue, unordered_set<int>& visited, vector<int>& currentPath, int currentWeight, int& bestWeight, vector<int>& bestPath, int startNodeData){
    if (visited.size() == nodes_map.size()){
        for (Edge* edge : nodes_map[currentNodeValue]->edges_to_node){
            if (edge->to->data == startNodeData){
                int totalCost = currentWeight + edge->weight;

                if (totalCost < bestWeight){
                    bestWeight = totalCost;
                    bestPath = currentPath;
                }
                break;
            }
        }
        return;
    }

    Node* currentNode = nodes_map[currentNodeValue];
```

```
    for (Edge* edge : currentNode->edges_to_node){
        if (visited.find(edge->to->data) == visited.end()){
            visited.insert(edge->to->data);
            currentPath.push_back(edge->to->data);

            TSPPathDecider(edge->to->data, visited, currentPath, currentWeight + edge->weight, bestWeight, bestPath, startNodeData);

            visited.erase(edge->to->data);
            currentPath.pop_back();
        }
    }
}
void MainWindow::TSPOperation(){
    int nodeStart = ui.TSPStart->text().toInt();

    vector<int>shortestPath = graph.TSPSolver(nodeStart);

    QString TSPResult;

    for (unsigned int i = 0; i < shortestPath.size(); i++){
        TSPResult.append(QString::number(shortestPath[i]));

        if (i < shortestPath.size() - 1){
            TSPResult.append(" -> ");
        }
    }
    static unsigned int idx = 0;

    // Смена активного узла при перемещении
    QTimer* timer = new QTimer(this);
    connect(timer, &QTimer::timeout, [=](){
        if (shortestPath.size() != 0 and idx < shortestPath.size()){
            Node* nod = graph.nodes_map[shortestPath[idx]];
            sNode = nod;
            nodeSelection = true;
            update();
            idx++;
        }else{
            timer->stop();
            timer->deleteLater();
            nodeSelection = false;
            update();
            idx = 0;
        }
    });

    ui.statusText->setText("Результат задачи коммивояджёра: " + TSPResult);
    timer->start(1000);
}

MainWindow::~MainWindow(){}
```

Код АРМ менеджера склада:

Mainwindow.h:

```h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <upgradeswindow.h>
#include <popup.h>
#include <cmath>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);

    ~MainWindow();

public slots:
    void receiveUpgrade(int itemType, int operation);
    void receiveMoneyData(int moneyData);
signals:
    void sendMoneyData(int);
    void sendEventData(QString, QString, QString);

private slots:
    void changeSellPercent();
    void updateDay();
    void sellVegetables();
    void sellFruits();
    void sellBerries();
    void openUpgrades();

private:
    Ui::MainWindow *ui;
    UpgradesWindow *uWindow;
    popup *eWindow;

    // Переменные
    int GameOver = 0;
    double DefectPercent = 0.4;
    int CurrentMoney = 0;
    int RentPay = 200;
    int RentTimer = 30;
    int Date[3] = {1, 1, 2000};
    double EventGainModifier = 1;
    double EventPriceModifier = 1;
    double EventDefectModifier = 1;

    // Методы
    void updateData();
    void Rent();
    void Drought();
    void Fertile();
    void Provider();
};

class Item{
public:
    int warehouseSize = 0;
    int warehouseLVL = 0;
    int stock = 0;
    int stockMax = 0;
    int gain = 0;
    double price = 0;
    double priceModifier = 1;
};

#endif // MAINWINDOW_H
```

Popup.h:

```cpp
#ifndef POPUP_H
#define POPUP_H

#include <QMainWindow>

namespace Ui {
class popup;
}

class popup : public QMainWindow
{
    Q_OBJECT

public:
    explicit popup(QWidget *parent = nullptr);
    ~popup();
public slots:
    void receiveEventData(QString eventName, QString eventDesc, QString eventEffects);
    void closePopup();
private:
    Ui::popup *ui;
};

#endif // POPUP_H
```

Upgradeswindow.h:

```h
#ifndef UPGRADESWINDOW_H
#define UPGRADESWINDOW_H

#include <QMainWindow>

namespace Ui {
class UpgradesWindow;
}

class UpgradesWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit UpgradesWindow(QWidget *parent = nullptr);
    ~UpgradesWindow();
public slots:
    void receiveMoneyData(int money);

signals:
    void sendUpgrade(int, int);
    void sendMoneyData(int);

private slots:
    void vegetablesExpand();
    void vegetablesUpgrade();
    void fruitsExpand();
    void fruitsUpgrade();
    void berriesExpand();
    void berriesUpgrade();
    void contractUpgrade();

private:
    Ui::UpgradesWindow *ui;

    // Переменные
    int money = 0;
    int vegetablesExpandCost = 1000;
    int fruitsExpandCost = 3000;
    int berriesExpandCost = 5000;
    int vegetablesUpgradeCost = 4000;
    int fruitsUpgradeCost = 6000;
    int berriesUpgradeCost = 8000;
    int contractUpgradeCost = 15000;

    // Методы

    void updateData();

};

#endif // UPGRADESWINDOW_H
```

Main.cpp:

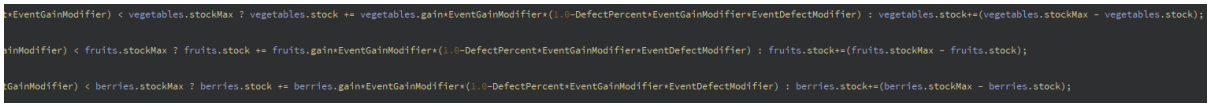
```cpp
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Mainwindow.cpp:

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QTimer>
#include <QRandomGenerator>

// Создание экземпляров класса товара
Item vegetables;
Item fruits;
Item berries;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // Инициализация окна улучшений и событий
    uWindow = new UpgradesWindow();
    eWindow = new popup();

    // Задание начального уровня и размера склада овощей
    vegetables.warehouseSize = 1;
    vegetables.warehouseLVL = 1;

    // Подключение слайдера к тексту % для продажи
    connect(ui->SellSlider, &QSlider::valueChanged, this, &MainWindow::changeSellPercent);

    // Продажа овощей
    connect(ui->SellVegetables, &QPushButton::pressed, this, &MainWindow::sellVegetables);
    // Продажа фркутов
    connect(ui->SellFruits, &QPushButton::pressed, this, &MainWindow::sellFruits);
    // Продажа ягод
    connect(ui->SellBerries, &QPushButton::pressed, this, &MainWindow::sellBerries);

    // Создание, подключение и активация таймера смены дня
    QTimer *DayTimer = new QTimer(this);
    connect(DayTimer, &QTimer::timeout, this, &MainWindow::updateDay);
    DayTimer->start(1000);

    // Подключение кнопки к открытию улучшений
    connect(ui->UpgradesBtn, &QPushButton::pressed, this, &MainWindow::openUpgrades);

    // Подключение сигналов передачи информации между окнами
    connect(uWindow, &UpgradesWindow::sendUpgrade, this, &MainWindow::receiveUpgrade);
    connect(uWindow, &UpgradesWindow::sendMoneyData, this, &MainWindow::receiveMoneyData);
    connect(this, &MainWindow::sendMoneyData, uWindow, &UpgradesWindow::receiveMoneyData);
    connect(this, &MainWindow::sendEventData, eWindow, &popup::receiveEventData);

    // Изначальный вызов обновления данных
    updateData();
}

// Обновление данных
```

```
// Обновление данных
void MainWindow::updateData(){
    // Обновление отображения количества товаров
    ui->Vegetables->setText(QString::number(vegetables.stock) + '/' + QString::number(vegetables.stockMax));
    ui->Fruits->setText(QString::number(fruits.stock) + '/' + QString::number(fruits.stockMax));
    ui->Berries->setText(QString::number(berries.stock) + '/' + QString::number(berries.stockMax));

    // Обновление отображения прироста
    ui->VegetablesDay->setText(QString::number(vegetables.gain));
    ui->FruitsDay->setText(QString::number(fruits.gain));
    ui->BerriesDay->setText(QString::number(berries.gain));

    // Обновление отображения цены
    ui->VegetablesPrice->setText(QString::number(round(vegetables.price*100)/100));
    ui->FruitsPrice->setText(QString::number(round(fruits.price*100)/100));
    ui->BerriesPrice->setText(QString::number(round(berries.price*100)/100));

    // Обновление процента брака
    ui->DefectPercent->setText(QString::number(round(DefectPercent * EventDefectModifier * 100*100)/100) + '%');

    // Обновление денег и следующего платежа
    ui->Money->setText(QString::number(CurrentMoney));
    ui->Rent->setText(QString::number(RentPay));

    // Обновление отображения даты
    QString dayString = QString::number(Date[0]);
    QString monthString = QString::number(Date[1]);
    QString yearString = QString::number(Date[2]);
    QString dateString = (Date[0] < 10 ? '0'+dayString : dayString) + '.' + (Date[1] < 10 ? '0'+monthString : monthString) + '.' +yearString;
    ui->Date->setText(dateString);

    // Обновление данных лимитов и прироста
    vegetables.stockMax = vegetables.warehouseSize * 1000;
    vegetables.gain = (vegetables.warehouseSize + vegetables.warehouseLVL * 2) * 20 * EventGainModifier;
    vegetables.priceModifier = 40 * pow(1.2, vegetables.warehouseLVL) * EventPriceModifier;

    fruits.stockMax = fruits.warehouseSize * 1000;
    fruits.gain = (fruits.warehouseSize + fruits.warehouseLVL * 2) * 20 * EventGainModifier;
    fruits.priceModifier = 1 * pow(1.2, fruits.warehouseLVL) * EventPriceModifier;

    berries.stockMax = berries.warehouseSize * 1000;
    berries.gain = (berries.warehouseSize + berries.warehouseLVL * 2) * 20 * EventGainModifier;
    berries.priceModifier = 1 * pow(1.2, berries.warehouseLVL) * EventPriceModifier;

    // Отправление данных о средствах в окно улучшений
    emit sendMoneyData(CurrentMoney);
}

// Смена дня
```

```
// Смена дня
void MainWindow::updateDay(){
    if (GameOver == 1){return;}
    // Прирост овощей
    vegetables.stock + vegetables.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier) < vegetables.stockMax ? vegetables.stock += vegetables.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier*EventDef

    // Прирост фруктов
    fruits.stock + fruits.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier) < fruits.stockMax ? fruits.stock += fruits.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier*EventDefectModifier) : fruit

    // Прирост ягод
    berries.stock + berries.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier) < berries.stockMax ? berries.stock += berries.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier*EventDefectModifier) :

    // Обновление даты
    if(Date[1] == 2){
        Date[0] < 28 ? Date[0]++ : (Date[0] = 1, (Date[1] < 12 ? Date[1]++ : (Date[1] = 1, Date[2]++)));
    } else if(Date[1]%2 == 0){
        Date[0] < 30? Date[0]++ : (Date[0] = 1, (Date[1] < 12 ? Date[1]++ : (Date[1] = 1, Date[2]++)));
    }else{
        Date[0] < 31? Date[0]++ : (Date[0] = 1, (Date[1] < 12 ? Date[1]++ : (Date[1] = 1, Date[2]++)));
    }

    // Обновление цен
    vegetables.price = (75 + QRandomGenerator::global()->bounded(75)) * vegetables.priceModifier * EventPriceModifier / 100.0;
    fruits.price = (175 + QRandomGenerator::global()->bounded(75)) * fruits.priceModifier * EventPriceModifier / 100.0;
    berries.price = (275 + QRandomGenerator::global()->bounded(75)) * berries.priceModifier * EventPriceModifier / 100.0;

    // Аренда и события
    if(RentTimer>1){
        RentTimer--;
    }else{
        if(RentPay > CurrentMoney){ Rent(); GameOver = 1; }else{
            CurrentMoney-=RentPay;
            EventDefectModifier = 1;
            EventGainModifier = 1;
            EventPriceModifier = 1;
            int eventChance = QRandomGenerator::global()->bounded(5);
            if (eventChance == 2){
                int eventType = QRandomGenerator::global()->bounded(3);
                if (eventType == 1){Drought();} else if (eventType == 2){ Fertile();} else if (eventType == 3){Provider();}
            }

            RentTimer = 30;
        }
    }

    // Вызов обновления данных
    updateData();
}

// Смена текста при движении слайдера
void MainWindow::changeSellPercent(){
    ui->SellPercent->setText(QString::number(ui->SellSlider->value())+'%');
}

// Продажа товара
void MainWindow::sellVegetables(){
    CurrentMoney += vegetables.stock*(ui->SellSlider->value() / 100.0) * vegetables.price;
    vegetables.stock -= vegetables.stock*(ui->SellSlider->value() / 100.0);

    updateData();
}
```



```
*EventGainModifier) < vegetables.stockMax ? vegetables.stock += vegetables.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier*EventDefectModifier) : vegetables.stock+=(vegetables.stockMax - vegetables.stock);

ainModifier) < fruits.stockMax ? fruits.stock += fruits.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier*EventDefectModifier) : fruits.stock+=(fruits.stockMax - fruits.stock);

GainModifier) < berries.stockMax ? berries.stock += berries.gain*EventGainModifier*(1.0-DefectPercent*EventGainModifier*EventDefectModifier) : berries.stock+=(berries.stockMax - berries.stock);
```

```
// Продажа товара
void MainWindow::sellVegetables(){
    CurrentMoney += vegetables.stock*(ui->SellSlider->value() / 100.0) * vegetables.price;
    vegetables.stock -= vegetables.stock*(ui->SellSlider->value() / 100.0);

    updateData();
}
void MainWindow::sellFruits(){
    CurrentMoney += fruits.stock*(ui->SellSlider->value() / 100.0) * fruits.price;
    fruits.stock -= fruits.stock*(ui->SellSlider->value() / 100.0);

    updateData();
}
void MainWindow::sellBerries(){
    CurrentMoney += berries.stock*(ui->SellSlider->value() / 100.0) * berries.price;
    berries.stock -= berries.stock*(ui->SellSlider->value() / 100.0);
    updateData();
}

// Открытие окна улучшений
void MainWindow::openUpgrades(){
    uWindow->show();
}

// Принятие данных о средствах из окна улучшений
void MainWindow::receiveMoneyData(int moneyData){
    CurrentMoney -= moneyData;
    updateData();
}

// Улучшение складов (1-3: Расширение/Улучшение/Контракт, 1-3: Овощи/Фрукты/Ягоды)
void MainWindow::receiveUpgrade(int operation, int itemType){
    if (operation == 1){
        if (itemType == 1){
            vegetables.warehouseSize++;
            RentPay = RentPay + 200 * vegetables.warehouseSize;
        }else if (itemType == 2){
            fruits.warehouseSize++;
            RentPay = RentPay + 300 * fruits.warehouseSize;
        }else if (itemType == 3){
            berries.warehouseSize++;
            RentPay = RentPay + 400 * berries.warehouseSize;
        }
    } else if(operation == 2){
        if (itemType == 1){
            vegetables.warehouseLVL++;
            RentPay = RentPay + 300 * vegetables.warehouseSize;
        }else if (itemType == 2){
            fruits.warehouseLVL++;
            RentPay = RentPay + 400 * fruits.warehouseSize;
        }else if (itemType == 3){
            berries.warehouseLVL++;
            RentPay = RentPay + 500 * berries.warehouseSize;
        }
    } else if (operation == 3){
        DefectPercent*=0.7;
        RentPay *= 1.2;
    }

    updateData();
}
```

Popup.cpp:

```cpp
#include "popup.h"
#include "ui_popup.h"

popup::popup(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::popup)
{
    ui->setupUi(this);

    connect(ui->ConfirmBTN, &QPushButton::pressed, this, &popup::closePopup);
}
void popup::receiveEventData(QString eventName, QString eventDesc, QString eventEffects){
    ui->EventName->setText(eventName);
    ui->EventDescription->setText(eventDesc);
    ui->Effects->setText(eventEffects);
}
void popup::closePopup(){
    this->close();
}
popup::~popup()
{
    delete ui;
}
```

```cpp
// События
void MainWindow::Rent(){
    emit sendEventData("Событие: Неуплата аренды.",
                       "Вы не смогли заплатить за аренду в этом месяце, в следствие чего ваш контракт на аренду склада был разорван.",
                       "Игра окончена.");
    eWindow->show();
}
void MainWindow::Drought(){
    int variant = QRandomGenerator::global()->bounded(2);
    if (variant == 1){
        emit sendEventData("Событие: Сезон засухи (глобальный).",
                           "Следующий месяц для всех будет скудным на урожай, в следствие чего урожай будет ниже, но цены на товары поднимутся.",
                           "Прирост товаров ниже, цена товаров выше.");
        EventGainModifier = 0.5; EventPriceModifier = 1.5;
    }else{
        emit sendEventData("Событие: Сезон засухи (локальный).",
                           "Следующий месяц для вас будет скудным на урожай, в следствие чего урожай будет ниже.",
                           "Прирост товаров ниже.");
        EventGainModifier = 0.5;
    }
    eWindow->show();
}
void MainWindow::Fertile(){
    int variant = QRandomGenerator::global()->bounded(2);
    if (variant == 1){
        emit sendEventData("Событие: Плодородный сезон (глобальный).",
                           "Следующий месяц для всех будет богатым на урожай, в следствие чего урожай будет выше, но цены на товары снизятся.",
                           "Прирост товаров выше, цена товаров ниже.");
        EventGainModifier = 1.5; EventPriceModifier = 0.5;
    }else{
        emit sendEventData("Событие: Плодородный сезон (локальный).",
                           "Следующий месяц для вас будет богатым на урожай, в следствие чего урожай будет выше.",
                           "Прирост товаров выше.");
        EventGainModifier = 1.5;
    }
    eWindow->show();
}
void MainWindow::Provider(){
    int variant = QRandomGenerator::global()->bounded(2);
    if (variant == 1){
        emit sendEventData("Событие: Проблемные поставки.",
                           "В результате некачественных переевозок, количество брака товара увеличилось.",
                           "Выше процент брака.");
        EventDefectModifier = 1.5;
    }else{
        emit sendEventData("Событие: Удачные поставки.",
                           "Вы смогли заключить удачный договор на следующий метод на высококачественный товар.",
                           "Ниже процент брака.");
        EventDefectModifier= 0.5;
    }
    eWindow->show();
}
MainWindow::~MainWindow()
{
    delete ui;
}
```

Upgradeswindow.cpp:

```cpp
#include "upgradeswindow.h"
#include "ui_upgradeswindow.h"

UpgradesWindow::UpgradesWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::UpgradesWindow)
{
    ui->setupUi(this);

    connect(ui->VegetablesBEUpgrade, &QPushButton::pressed, this, &UpgradesWindow::vegetablesExpand);
    connect(ui->VegetablesUpgrade, &QPushButton::pressed, this, &UpgradesWindow::vegetablesUpgrade);

    connect(ui->FruitsBEUpgrade, &QPushButton::pressed, this, &UpgradesWindow::fruitsExpand);
    connect(ui->FruitsUpgrade, &QPushButton::pressed, this, &UpgradesWindow::fruitsUpgrade);

    connect(ui->BerriesBEUpgrade, &QPushButton::pressed, this, &UpgradesWindow::berriesExpand);
    connect(ui->BerriesUpgrade, &QPushButton::pressed, this, &UpgradesWindow::berriesUpgrade);

    connect(ui->ContractUpgrade, &QPushButton::pressed, this, &UpgradesWindow::contractUpgrade);

    updateData();
}

// Приём количества средств
void UpgradesWindow::receiveMoneyData(int moneyData){
    money = moneyData;
}

// Улучшение складов (1-3: Расширение/Улучшение/Контракт, 1-3: Овощи/Фрукты/Ягоды)
void UpgradesWindow::vegetablesExpand(){
    if( money >= vegetablesExpandCost){ emit sendUpgrade(1,1); emit sendMoneyData(vegetablesExpandCost); vegetablesExpandCost*=1.3; updateData();}
}
void UpgradesWindow::fruitsExpand(){
    if( money >= fruitsExpandCost){ emit sendUpgrade(1,2); emit sendMoneyData(fruitsExpandCost); fruitsExpandCost*=1.3; updateData();}
}
void UpgradesWindow::berriesExpand(){
    if( money >= berriesExpandCost){ emit sendUpgrade(1,3); emit sendMoneyData(berriesExpandCost); berriesExpandCost*=1.3; updateData();}
}
void UpgradesWindow::vegetablesUpgrade(){
    if( money >= vegetablesUpgradeCost){ emit sendUpgrade(2,1); emit sendMoneyData(vegetablesUpgradeCost); vegetablesUpgradeCost*=1.4; updateData();}
}
void UpgradesWindow::fruitsUpgrade(){
    if( money >= fruitsUpgradeCost){ emit sendUpgrade(2,2); emit sendMoneyData(fruitsUpgradeCost); fruitsUpgradeCost*=1.4; updateData();}
}
void UpgradesWindow::berriesUpgrade(){
    if( money >= berriesUpgradeCost){ emit sendUpgrade(2,3); emit sendMoneyData(berriesUpgradeCost); berriesUpgradeCost*=1.4; updateData();}
}
void UpgradesWindow::contractUpgrade(){
    if( money >= contractUpgradeCost){ emit sendUpgrade(3,0); emit sendMoneyData(contractUpgradeCost); contractUpgradeCost*=1.5; updateData();}
}

void UpgradesWindow::updateData(){
    ui->VegetablesBEUpgrade->setText("Расширить склад\nовощей\n" + QString::number(vegetablesExpandCost) + '$');
    ui->FruitsBEUpgrade->setText("Расширить склад\nфруктов\n" + QString::number(fruitsExpandCost) + '$');
    ui->BerriesBEUpgrade->setText("Расширить склад\nягод\n" + QString::number(berriesExpandCost) + '$');

    ui->VegetablesUpgrade->setText("Улучшить склад\nовощей\n" + QString::number(vegetablesUpgradeCost) + '$');
    ui->FruitsUpgrade->setText("Улучшить склад\nфруктов\n" + QString::number(fruitsUpgradeCost) + '$');
    ui->BerriesUpgrade->setText("Улучшить склад\nягод\n" + QString::number(berriesUpgradeCost) + '$');

    ui->ContractUpgrade->setText("Сменить контракт на\nболее выгодный\n" + QString::number(contractUpgradeCost) + '$');
}

UpgradesWindow::~UpgradesWindow(){ delete ui; }
```