**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. №подл |  |

СОГЛАСОВАНО

Научный руководитель, к.т.н., доцент ДПИ ФКН

\_\_\_\_\_\_\_\_\_ Р.А. Родригес Залепинос

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия», к.т.н., профессор ДПИ ФКН

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**ПРОГРАММА МОДЕЛИРОВАНИЯ РАСПРОСТРАНЕНИЯ ПОЖАРА С ПОМОЩЬЮ КЛЕТОЧНЫХ АВТОМАТОВ**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.05.06-01 12 01-1-ЛУ**

**Исполнитель**

Студент группы БПИ196

\_\_\_\_\_\_\_\_\_\_\_\_ / М. И. Филиппова/

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Москва 2021**

**УТВЕРЖДЕН**

**RU.17701729.05.06-01 12 01-1 ЛУ**

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. №подл |  |

**ПРОГРАММА МОДЕЛИРОВАНИЯ РАСПРОСТРАНЕНИЯ ПОЖАРА С ПОМОЩЬЮ КЛЕТОЧНЫХ АВТОМАТОВ**

**Текст программы**

**RU.17701729.05.06-01 12 01-1**

**Листов 117**

**Москва 2021**

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc83368380)

[**1.** **Подсистема модели** 4](#_Toc83368381)

[**1.1** **Пакет forest** 4](#_Toc83368382)

[**1.1.1** **Класс ForestArea** 4](#_Toc83368383)

[**1.1.2** **Класс ForestCell** 24](#_Toc83368384)

[**1.1.3** **Класс ForestStates** 34](#_Toc83368385)

[**1.3** **Пакет urban** 39](#_Toc83368386)

[**1.3.1** **Класс UrbanArea** 39](#_Toc83368387)

[**1.3.2** **Класс UrbanCell** 56](#_Toc83368388)

[**1.3.3** **Класс UrbanStates** 64](#_Toc83368389)

[**1.4** **Класс GlobalFire** 65](#_Toc83368390)

[**1.5** **Класс Main** 77](#_Toc83368391)

[**2.** **Подсистема визуализации** 79](#_Toc83368392)

[**2.1** **Файл UIWindow.py** 79](#_Toc83368393)

[**2.2** **Файл UIElements** 85](#_Toc83368394)

[**2.3** **Файл MapView.py** 98](#_Toc83368395)

[**2.4** **Файл MapEngine.py** 104](#_Toc83368396)

[**2.5** **Файл Layers.py** 109](#_Toc83368397)

[**2.6** **Файл Utilities.py** 110](#_Toc83368398)

[**2.7** **Файл Layers.py** 114](#_Toc83368399)

[**2.8** **Файл Exceptions.py** 115](#_Toc83368400)

[**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ** 117](#_Toc83368401)

# **ВВЕДЕНИЕ**

Программа состоит из модели и подсистемы визуализации. Далее будет приведен код подсистем.

1. **Подсистема модели**
   1. **Пакет forest**
      1. **Класс ForestArea**

**package** com.model.forest;  
  
**import** com.model.urban.UrbanStates;  
**import** com.opencsv.CSVReader;  
**import** com.opencsv.exceptions.CsvValidationException;  
**import** org.apache.commons.lang3.ArrayUtils;  
**import** org.gdal.gdal.\*;  
**import** org.gdal.gdalconst.gdalconst;  
**import** org.gdal.ogr.DataSource;  
**import** org.gdal.ogr.ogr;  
**import** org.gdal.ogr.ogrConstants;  
**import** org.gdal.osr.CoordinateTransformation;  
**import** org.gdal.osr.SpatialReference;  
  
**import** java.io.File;  
**import** java.io.FileReader;  
**import** java.io.IOException;  
**import** java.time.LocalDateTime;  
**import** java.util.\*;  
**import** java.util.stream.Collectors;  
**import** java.util.stream.Stream;  
  
**import** com.model.input.InputData;  
**import** com.model.urban.UrbanCell;  
  
**public class** ForestArea {  
 **int side**;  
 **int length**, **width**;  
 ForestCell[][] **cells**;  
 InputData **inputData**;  
 String **ignitionRasterPath** = **"../data/ignition/ignition.tif"**;  
  
 **public void** setLength(**int** length) {  
 **this**.**length** = length;  
 }  
  
 **public void** setWidth(**int** width) {  
 **this**.**width** = width;  
 }  
  
 **public void** setSpatialReferenceUTM(SpatialReference spatialReferenceUTM) {  
 **this**.**spatialReferenceUTM** = spatialReferenceUTM;  
 }  
  
 LocalDateTime **currentDate**;  
  
 ForestStates[][] **states**;  
  
 **private** SpatialReference **spatialReferenceUTM**;  
  
 **public** ForestArea(InputData inputData, SpatialReference spatialReferenceUTM, **int** length, **int** width) {  
 **this**.**side** = inputData.getSide();  
 **this**.**inputData** = inputData;  
 **currentDate** = inputData.getStart();  
 ForestCell.*setSide*(**side**);  
 **this**.**spatialReferenceUTM** = spatialReferenceUTM;  
 **this**.**length** = length;  
 **this**.**width** = width;  
  
 defineArea(inputData);  
 defineNeighbours();  
  
 gdal.*AllRegister*();  
 setElevation(inputData.getElevation());  
 setSlopes();  
 setFuel(inputData.getFuel(), inputData.getFuelCodes());  
 setIgnition(inputData.getIgnition());  
  
 }  
  
 **private void** setSpreadRates() {  
 **for** (**int** i = 1; i < **width** - 1; i++) {  
 **for** (**int** j = 1; j < **length** - 1; j++) {  
 **cells**[i][j].initSpreadRates();  
 }  
 }  
 }  
  
  
 **private void** defineArea(InputData inputData) {  
 **cells** = **new** ForestCell[**width**][**length**];  
  
 **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** transform = **new** CoordinateTransformation(sourceSRS, **spatialReferenceUTM**);  
  
 **double**[] start = transform.TransformPoint(inputData.getStartPoint().GetX(),  
 inputData.getStartPoint().GetY());  
  
 **double** x, y;  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 x = start[1] + i \* **side**;  
 y = start[0] + j \* **side**;  
 **cells**[i][j] = **new** ForestCell(x, y);  
 }  
 }  
  
 **states** = **new** ForestStates[**width**][**length**];  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **states**[i][j] = ForestStates.***UNBURNED***;  
 }  
 }  
 }  
  
 **private void** setIgnition(String ignition) {  
 DataSource ign = ogr.*Open*(ignition);  
  
 **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** transform = **new** CoordinateTransformation(sourceSRS, **spatialReferenceUTM**);  
  
 **double**[] start = transform.TransformPoint(**inputData**.getStartPoint().GetX(),  
 **inputData**.getStartPoint().GetY());  
  
 **for** (**int** i = 0; i < ign.GetLayerCount(); i++) {  
 **for** (**int** j = 0; j < ign.GetLayer(i).GetFeatureCount(); j++) {  
  
 **var** geom = ign.GetLayer(i).GetFeature(j).GetGeometryRef();  
 **var** source = geom.GetSpatialReference();  
 **var** trans = **new** CoordinateTransformation(source, **spatialReferenceUTM**);  
 **double**[] point;  
  
 **if** (geom.GetGeometryType() == ogrConstants.***wkbPoint***) {  
 point = trans.TransformPoint(geom.GetX(), geom.GetY());  
 **int** y = (**int**) Math.*round*(Math.*abs*(point[1] - start[1]) / **side**),  
 x = (**int**) Math.*round*(Math.*abs*(point[0] - start[0]) / **side**);  
  
 **cells**[x][y].setState(ForestStates.***DEVELOPING***);  
 **states**[x][y] = ForestStates.***DEVELOPING***;  
 }  
  
 **if** (geom.GetGeometryType() == ogrConstants.***wkbPolygon***) {  
 rasterizeIgnition(ignition);  
 **var** dataset = gdal.*Open*(**ignitionRasterPath**);  
 **var** band = dataset.GetRasterBand(1);  
 **int**[] presence = **new int**[1];  
  
 **for** (**int** k = 0; i < **width**; i++) {  
 **for** (**int** l = 0; j < **length**; j++) {  
 band.ReadRaster(k, **length** - 1 - l, 1, 1, presence);  
 **if** (presence[0] > 0) {  
 **states**[k][l] = ForestStates.***DEVELOPING***;  
 }  
 }  
 }  
  
 band.delete();  
 dataset.delete();  
 }  
 }  
 }  
 }  
  
 **private void** rasterizeIgnition(String ignition) {  
 **var** ignitionData = ogr.*Open*(ignition);  
 **var** ignitionLayer = ignitionData.GetLayer(0);  
  
 SpatialReference sourceSrs = ignitionLayer.GetSpatialRef();  
 **double**[] extent = ignitionLayer.GetExtent();  
  
 **double** x\_res = ((extent[1] - extent[0]) / **side**);  
 **double** y\_res = ((extent[3] - extent[2]) / **side**);  
  
 **int** xCor = (**int**) x\_res;  
 **int** yCor = (**int**) y\_res;  
  
 Dataset target\_ds = gdal.*GetDriverByName*(**"GTiff"**)  
 .Create(**ignitionRasterPath**, xCor, yCor, 1, gdalconst.***GDT\_Byte***);  
 target\_ds.SetProjection(sourceSrs.ExportToPrettyWkt());  
 target\_ds.SetGeoTransform(**new double**[]{extent[0], **side**, 0, extent[3], 0, -**side**});  
 Band band = target\_ds.GetRasterBand(1);  
  
  
 **int**[] intArr = {1};  
  
 *// Rasterize* gdal.*RasterizeLayer*(target\_ds, intArr, ignitionLayer, **null**);  
  
 ignitionLayer.delete();  
 ignitionData.delete();  
 target\_ds.delete();  
 band.delete();  
 }  
  
 **private void** defineNeighbours() {  
 **for** (**int** i = 1; i < **width** - 1; i++) {  
 **for** (**int** j = 1; j < **length** - 1; j++) {  
 **cells**[i][j].setNeighbours(**new** ForestCell[]{**cells**[i - 1][j], **cells**[i - 1][j + 1],  
 **cells**[i][j + 1], **cells**[i + 1][j + 1],  
 **cells**[i + 1][j], **cells**[i + 1][j - 1], **cells**[i][j - 1], **cells**[i - 1][j - 1]});  
 }  
 }  
 }  
  
 **public void** propagate(**double** minutesLeft, **double** step, LocalDateTime localDateTime) {  
 **double** newState = 0;  
 **currentDate** = localDateTime;  
 *// поменять погоду* **if** (minutesLeft == 0) {  
 setSpreadRates();  
 }  
  
 **for** (**int** i = 2; i < **width** - 2; i++) {  
 **for** (**int** j = 2; j < **length** - 2; j++) {  
  
 **switch** (**cells**[i][j].getState()) {  
 **case *UNBURNED***:  
 **if** (Arrays.*stream*(**cells**[i][j].**neighbours**)  
 .anyMatch(x -> x.getState().equals(ForestStates.***DEVELOPING***))) {  
  
 newState = (**cells**[i - 1][j - 1].getSpreadRates()[3] +  
 **cells**[i + 1][j - 1].getSpreadRates()[5] +  
 **cells**[i - 1][j + 1].getSpreadRates()[1] +  
 **cells**[i + 1][j + 1].getSpreadRates()[7]) \* **cells**[i][j].getFirePeriod() /  
 Math.*sqrt*(2) / **side** + **cells**[i][j].getState().getValue() +  
 (**cells**[i][j - 1].getSpreadRates()[4] +  
 **cells**[i - 1][j].getSpreadRates()[2] +  
 **cells**[i + 1][j].getSpreadRates()[6] +  
 **cells**[i][j + 1].getSpreadRates()[0]) \* **cells**[i][j].getFirePeriod() / **side**;  
  
 **if** (newState >= 1)  
 **states**[i][j] = ForestStates.***IGNITED***;  
 }  
  
 **break**;  
 **case *IGNITED***:  
 **if** (**cells**[i][j].getInnerFireTime() == 0) {  
 **var** time = **side** / (Math.*sqrt*(Math.***PI***) \* **cells**[i][j].calculateInternalSpreadRate());  
 **cells**[i][j].setInnerFireTime(time \* 60);  
 } **else** {  
 **cells**[i][j].setInnerFireTime(Math.*max*(0.0, **cells**[i][j].getInnerFireTime() - step));  
 **if** (**cells**[i][j].getInnerFireTime() == 0)  
 **states**[i][j] = ForestStates.***DEVELOPING***;  
 }  
 **break**;  
 **case *DEVELOPING***:  
 **if** (**cells**[i][j].getInnerFireTime() == 0) {  
 **cells**[i][j].setInnerFireTime(**cells**[i][j].getFirePeriod() \* 60);  
 } **else** {  
 **cells**[i][j].setInnerFireTime(Math.*max*(0, **cells**[i][j].getInnerFireTime() - step));  
 **if** (**cells**[i][j].getInnerFireTime() == 0)  
 **states**[i][j] = ForestStates.***EXTINGUISHING***;  
 }  
 **break**;  
 **case *EXTINGUISHING***:  
 **if** (**cells**[i][j].getInnerFireTime() == 0) {  
 **cells**[i][j].setInnerFireTime(**cells**[i][j].getFirePeriod() \* 60);  
 } **else** {  
 **cells**[i][j].setInnerFireTime(Math.*max*(0, **cells**[i][j].getInnerFireTime() - step));  
 **if** (**cells**[i][j].getInnerFireTime() == 0)  
 **states**[i][j] = ForestStates.***BURNED***;  
 }  
 **break**;  
 **case *BURNED***:  
 **break**;  
 }  
 }  
 }  
 }  
  
  
 **public void** printStatistics() {  
 **int** ignited = 0;  
 **int** burned = 0;  
 **int** developing = 0;  
 **int** unb = 0;  
 **int** ext = 0;  
  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **switch** (**cells**[i][j].getState()) {  
 **case *UNBURNED*** -> {  
 unb++;  
 }  
 **case *IGNITED*** -> {  
 ignited++;  
 }  
 **case *DEVELOPING*** -> {  
 developing++;  
 }  
 **case *EXTINGUISHING*** -> {  
 ext++;  
 }  
 **case *BURNED*** -> {  
 burned++;  
 }  
 }  
 }  
 }  
  
 System.***out***.println(**"========"** + **currentDate**.toString() + **"========"**);  
 System.***out***.println(**"IGNITED = "** + ignited);  
 System.***out***.println(**"BURNED = "** + burned);  
 System.***out***.println(**"DEVELOPING = "** + developing);  
 System.***out***.println(**"UNBURNED = "** + unb);  
 System.***out***.println(**"EXTINGUISHING = "** + ext);  
 }  
  
 **public void** updateStates() {  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **if** (**cells**[i][j].isIgnitedByUrban()) {  
 **cells**[i][j].setState(ForestStates.***IGNITED***);  
 **cells**[i][j].makeIgnitedByUrbanDefault();  
 **states**[i][j] = ForestStates.***IGNITED***;  
 }  
 **cells**[i][j].setState(**states**[i][j]);  
 }  
 }  
 }  
  
 **public void** setFuel(String path, String fuelCodes) {  
 Map<String, Double> fuelTypesTransition = Stream.*of*(**new** Object[][]{  
 {**"Tree"**, 0.6},  
 {**"Shrub"**, 0.7},  
 {**"Herb"**, 0.3},  
 {**"Agriculture"**, 1.2},  
 {**"Sparse"**, 0.1} *// Barren, Water, Snow-Ice, NA -> 0* }).collect(Collectors.*toMap*(data -> (String) data[0], data -> (Double) data[1]));  
  
 *// read fuel Codes* Map<Integer, Double> codes = readFuelCodes(fuelCodes, fuelTypesTransition);  
  
 Dataset fuel = gdal.*Open*(path);  
  
 **var** paths = generatePaths(path, **"fuel.tif"**);  
  
 fuel = changeProjection(fuel, paths[0]);  
 Dataset modified = changeResolutionAndBorders(fuel, paths[1]);  
  
 Band fuelTypes = modified.GetRasterBand(1);  
 **int**[] value = **new int**[1];  
  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 fuelTypes.ReadRaster(i, **length** - 1 - j, 1, 1, value);  
  
 **var** val = codes.getOrDefault(value[0], 0.0) != **null** ?  
 codes.getOrDefault(value[0], 0.0) : 0.0;  
 **cells**[i][j].setFuel(val);  
 }  
 }  
  
 fuel.delete();  
 modified.delete();  
 }  
  
 **private** Map<Integer, Double> readFuelCodes(String fuelCodes, Map<String, Double> fuelTypesTransition) {  
 Map<Integer, Double> codes = **new** HashMap<>();  
 **try** {  
 **var** fileReader = **new** FileReader(fuelCodes);  
 **var** csvReader = **new** CSVReader(fileReader);  
 String[] record;  
  
 *// Заголовок.* record = csvReader.readNext();  
 **int** index = ArrayUtils.*indexOf*(record, **"EVT\_LF"**);  
  
 **while** ((record = csvReader.readNext()) != **null**) {  
 codes.put(Integer.*valueOf*(record[0]), fuelTypesTransition.get(record[index]));  
 }  
  
 csvReader.close();  
 fileReader.close();  
 }  
 **catch** (CsvValidationException | IOException e) {  
 e.printStackTrace();  
 }  
 **return** codes;  
 }  
  
 **public void** setWeatherData(String weatherDataPath) {  
 **var** dataset = gdal.*Open*(weatherDataPath);  
 Band velocity = dataset.GetRasterBand(1);  
 Band angle = dataset.GetRasterBand(2);  
 Band temperature = dataset.GetRasterBand(3);  
 Band humidity = dataset.GetRasterBand(4);  
  
 **var** temp = **new double**[1];  
 **var** hum = **new double**[1];  
 **var** vel = **new double**[1];  
 **var** ang = **new double**[1];  
  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 temperature.ReadRaster(i, j, 1, 1, temp);  
 humidity.ReadRaster(i, j, 1, 1, hum);  
 velocity.ReadRaster(i, j, 1, 1, vel);  
 angle.ReadRaster(i, j, 1, 1, ang);  
  
 **cells**[i][j].changeDefaultSpreadRate(temp[0],  
 vel[0], hum[0]);  
 **cells**[i][j].setWindDirection(ang[0]);  
 **cells**[i][j].setWindVelocity(vel[0]);  
  
 }  
 }  
  
 temperature.delete();  
 humidity.delete();  
 velocity.delete();  
 angle.delete();  
 dataset.delete();  
 }  
  
 **private** String[] generatePaths(String path, String name) {  
 **var** ind = path.lastIndexOf(File.***separator***);  
 **var** projectedPath = path.substring(0, ind + 1) + **"projected\_"** + name;  
 **var** modifiedPath = path.substring(0, ind + 1) + **"modified\_"** + name;  
 **return new** String[]{projectedPath, modifiedPath};  
  
 }  
  
 **public void** setElevation(String path) {  
 Dataset elevation = gdal.*Open*(path);  
  
 **var** paths = generatePaths(path, **"elevation.tif"**);  
  
 elevation = changeProjection(elevation, paths[0]);  
 Dataset modified = changeResolutionAndBorders(elevation, paths[1]);  
  
 Band heights = modified.GetRasterBand(1);  
  
 **double**[] v = **new double**[2];  
 heights.ComputeRasterMinMax(v);  
 **int** mean = (**int**) Math.*round*((v[0] + v[1]) / 2);  
  
 Double[] noDataValue = **new** Double[1];  
 heights.GetNoDataValue(noDataValue);  
  
 **int**[] value = **new int**[1];  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 heights.ReadRaster(i, **length** - 1 - j, 1, 1, value);  
 **if** (value[0] == noDataValue[0]) value[0] = j > 1 ?  
 (**int**) **cells**[i][j - 1].getHeight() : mean;  
 **cells**[i][j].setHeight(value[0]);  
  
 }  
 }  
 elevation.delete();  
 modified.delete();  
 }  
  
 **private void** setSlopes() {  
 **for** (**int** i = 1; i < **width** - 1; i++) {  
 **for** (**int** j = 1; j < **length** - 1; j++) {  
 **cells**[i][j].initSlope();  
 }  
 }  
 }  
  
  
 **private** Dataset changeResolutionAndBorders(Dataset dataset, String path) {  
 *// Изменить размер и разрешение* **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** targetSRS = dataset.GetSpatialRef();  
  
 **var** ct = **new** CoordinateTransformation(sourceSRS, targetSRS);  
  
 **var** beginning = ct.TransformPoint(**inputData**.getStartPoint().GetX(), **inputData**.getStartPoint().GetY());  
 **var** finish = ct.TransformPoint(**inputData**.getEndPoint().GetX(), **inputData**.getEndPoint().GetY());  
  
 Vector<String> options =  
 **new** Vector<>(Arrays.*asList*(**"-te"**, String.*valueOf*(beginning[0]), String.*valueOf*(beginning[1]),  
 String.*valueOf*(finish[0]), String.*valueOf*(finish[1]),  
 **"-tr"**, String.*valueOf*(**side**), String.*valueOf*(**side**)));  
  
 **var** warpOptions = **new** WarpOptions(options);  
 Dataset[] srcData = {dataset};  
 Dataset modified = gdal.*Warp*(path, srcData, warpOptions);  
 **return** modified;  
 }  
  
 **private** Dataset changeProjection(Dataset dataset, String path) {  
 Vector<String> options = **new** Vector<>();  
 options.add(**"-t\_srs"**);  
 options.add(**spatialReferenceUTM**.ExportToPrettyWkt());  
 WarpOptions warpOptions = **new** WarpOptions(options);  
 Dataset[] srcData = {dataset};  
 Dataset projected = gdal.*Warp*(path, srcData, warpOptions);  
 dataset = gdal.*Open*(path);  
  
 **return** dataset;  
 }  
  
  
 **public** ForestCell[][] getCells() {  
 **return cells**;  
 }  
  
 **public void** propagateInUrban(UrbanCell[][] urbanCells) {  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **if** (**cells**[i][j].getState().equals(ForestStates.***DEVELOPING***)) {  
 **cells**[i][j].fireSpreadOnUrban(urbanCells, i, j, **width**, **length**);  
  
 }  
 }  
  
 }  
 }  
}

* + 1. **Класс ForestCell**

**package** com.model.forest;  
  
**import** org.gdal.ogr.Geometry;  
**import** org.gdal.ogr.ogr;  
**import** com.model.urban.UrbanCell;  
**import** com.model.urban.UrbanStates;  
  
**import** java.util.Arrays;  
  
**public class** ForestCell {  
 String **geometry**;  
 **private boolean ignitedByUrban** = **false**;  
 ForestStates **state**;  
 **private double innerFireTime**;  
 **double maxSpreadRate** = 0.0;  
 **double firePeriod** = 0.0;  
 **double fuel**;  
 **double windVelocity**;  
 **double windDirection**;  
 **double height**;  
 **double spreadRateDefault**;  
 ForestCell[] **neighbours**; *// N NE E SE S SW W NW* **double slope**;  
 **double**[] **spreadRates**;  
 **static int** *side*;  
  
  
 **public boolean** isIgnitedByUrban() {  
 **return ignitedByUrban**;  
 }  
  
 **public** String getGeometry() {  
 **return geometry**;  
 }  
  
 **public** ForestCell(**double** x, **double** y) {  
 **state** = ForestStates.***UNBURNED***;  
 Geometry poly = calculateGeometry(x, y);  
 **this**.**geometry** = poly.ExportToWkt();  
 }  
  
 **private** Geometry calculateGeometry(**double** x, **double** y) {  
 **var** ring = **new** Geometry(ogr.***wkbLinearRing***);  
 ring.AddPoint(x, y);  
 ring.AddPoint(x + *side*, y);  
 ring.AddPoint(x + *side*, y + *side*);  
 ring.AddPoint(x, y + *side*);  
 ring.AddPoint(x, y);  
  
 **var** poly = **new** Geometry(ogr.***wkbPolygon***);  
 poly.AddGeometry(ring);  
 **return** poly;  
 }  
  
 **public** ForestStates getState() {  
 **return state**;  
 }  
  
 **public void** setState(ForestStates state) {  
 **this**.**state** = state;  
 }  
  
 **public double** getMaxSpreadRate() {  
 **return maxSpreadRate**;  
 }  
  
 **public double** getFirePeriod() {  
 **return firePeriod**;  
 }  
  
 **public void** setFuel(**double** fuel) {  
 **this**.**fuel** = fuel;  
 }  
  
 **public double** getFuel() {  
 **return fuel**;  
 }  
  
  
 **public void** setWindVelocity(**double** windVelocity) {  
 **this**.**windVelocity** = windVelocity;  
 }  
  
 **public void** setWindDirection(**double** windDirection) {  
 **this**.**windDirection** = windDirection;  
 }  
  
 **public double** getWindVelocity() {  
 **return windVelocity**;  
 }  
  
 **public void** setHeight(**double** height) {  
 **this**.**height** = height;  
 }  
  
 **public double**[] getSpreadRates() {  
 **return spreadRates**;  
 }  
  
 **public void** setNeighbours(ForestCell[] neighbours) {  
 **this**.**neighbours** = neighbours;  
 }  
  
 **public static int** getSide() {  
 **return** *side*;  
 }  
  
 **public static void** setSide(**int** side) {  
 ForestCell.*side* = side;  
 }  
  
 **public double** getHeight() {  
 **return height**;  
 }  
  
 **public double** getWindDirection() {  
 **return windDirection**;  
 }  
  
 **public void** initSlope() {  
 **var** x = Math.*ceil*((**neighbours**[1].getHeight() - **neighbours**[7].getHeight())  
 + 2 \* (**neighbours**[2].getHeight() - **neighbours**[6].getHeight())  
 + (**neighbours**[3].getHeight() - **neighbours**[5].getHeight())) /  
 (8 \* *side*); *//(neighbours[2].getHeight() - neighbours[6].getHeight()) / (2 \* side);* **var** y = Math.*ceil*((**neighbours**[7].getHeight() - **neighbours**[5].getHeight())  
 + 2 \* (**neighbours**[0].getHeight() - **neighbours**[4].getHeight())  
 + (**neighbours**[1].getHeight() - **neighbours**[3].getHeight())) /  
 (8 \* *side*); *//(neighbours[0].getHeight() - neighbours[4].getHeight()) / (2 \* side);* **slope** = Math.*toDegrees*(Math.*atan*(Math.*sqrt*(x \* x + y \* y)));  
 }  
  
 **public void** changeDefaultSpreadRate(**double** temperature,  
 **double** windVelocity,  
 **double** humidity) {  
 **this**.**spreadRateDefault** = 0.03 \* temperature + 0.05 \* windVelocity +  
 0.01 \* (100 - humidity) - 0.3;  
  
 }  
  
 **public void** initSpreadRates() {  
 **if** (**spreadRates** == **null**)  
 **spreadRates** = **new double**[8];  
 **for** (**int** i = 0; i < 8; i++) {  
 **spreadRates**[i] = calculateSpreadRate(i);  
 }  
 **maxSpreadRate** = Arrays.*stream*(**spreadRates**).max().getAsDouble();  
 **if** (**maxSpreadRate** > 0) **firePeriod** = 0.45 \* *side* / **maxSpreadRate**;  
  
 }  
  
  
 **private double** calculateSpreadRate(**int** i) {  
 *// N NE E SE S SW W NW* **double** wind = **switch** (i) {  
 **case** 0 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection** - 180)));  
 **case** 1 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection** - 135)));  
 **case** 2 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection** - 90)));  
 **case** 3 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection** - 225)));  
 **case** 4 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection**)));  
 **case** 5 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection** + 45)));  
 **case** 6 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection** + 90)));  
 **case** 7 -> Math.*exp*(0.1783 \* **windVelocity** \* Math.*cos*(Math.*toRadians*(**windDirection** + 225)));  
 **default** -> 1;  
 };  
  
 **int** sign = 1;  
 *//if (getHeight() > neighbours[i].getHeight()) sign = -1;* **double** sl = Math.*exp*(sign \* 3.533 \* Math.*pow*(Math.*tan*(Math.*toRadians*(**slope**)  
 \* Math.*abs*(Math.*cos*(Math.*toRadians*(**windDirection**)))), 1.2));  
  
  
 **return spreadRateDefault** \* **fuel** \* wind \* sl;  
  
 }  
  
 **public double** calculateInternalSpreadRate() {  
  
 **int** sign = 1;  
 **double** sl = Math.*exp*(sign \* 3.533 \*  
 Math.*pow*(Math.*tan*(Math.*toRadians*(**slope**)  
 \* Math.*abs*(Math.*cos*(Math.*toRadians*(**windDirection**)))), 1.2));  
  
 **return spreadRateDefault** \* **fuel** \* sl \* Math.*exp*(0.1783 \* **windVelocity**);  
 }  
  
 **public double** getInnerFireTime() {  
 **return innerFireTime**;  
 }  
  
 **public void** setInnerFireTime(**double** innerFireTime) {  
 **this**.**innerFireTime** = innerFireTime;  
 }  
  
  
 **public void** fireSpreadOnUrban(UrbanCell[][] urbanCells, **int** i, **int** j, **int** width, **int** length) {  
 **var** k = getMaxSpreadRate() < 13.1 ? 3 : 4.5;  
 **var** a = (3 \* getWindVelocity() / 5 + 3) \* k + *side* \* 1.0 / 2;  
 **var** b = -2 \* getWindVelocity() / 15 + 3 + *side* \* 1.0 / 2;  
 **var** c = -1 \* getWindVelocity() / 15 + 3 + *side* \* 1.0 / 2;  
  
 **var** t = Math.*sqrt*(b \* (a + c) / 2.0);  
 **var** geom = Geometry.*CreateFromWkt*(**geometry**).Centroid();  
 **double** x = geom.GetX(), y = geom.GetY();  
 **var** influence = **new** Geometry(ogr.***wkbLinearRing***);  
 **double**[] f = rotatedCoords(x - t, y + c, x, y, **windDirection**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x + t, y + c, x, y, **windDirection**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x + t, y - a, x, y, **windDirection**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x - t, y - a, x, y, **windDirection**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x - t, y + c, x, y, **windDirection**);  
 influence.AddPoint(f[0], f[1]);  
  
 **var** poly = **new** Geometry(ogr.***wkbPolygon***);  
 poly.AddGeometry(influence);  
  
  
 **int** mini = (**int**) Math.*max*(0, i - a / *side*);  
 **int** minj = (**int**) Math.*max*(0, j - a / *side*);  
 **int** maxi = (**int**) Math.*min*(width - 1, i + a / *side*);  
 **int** maxj = (**int**) Math.*min*(length - 1, j + a / *side*);  
  
 **double** ign;  
 **for** (**int** l = mini; l <= maxi; l++) {  
 **for** (**int** m = minj; m <= maxj; m++) {  
 **if** (urbanCells[l][m] != **null** && urbanCells[l][m].getState().equals(UrbanStates.***UNBURNED***)) {  
 **var** urbanGeom = Geometry.*CreateFromWkt*(urbanCells[l][m].getGeometry());  
 **if** (urbanGeom.Intersects(poly)) {  
 ign = urbanCells[l][m].getMaterial() \* urbanCells[l][m].getWeather()  
 \* urbanGeom.Intersection(poly).Area() / urbanGeom.Area();  
 **if** (k == 3)  
 ign \*= 0.4;  
 urbanCells[l][m].addIgnitionProbability(1 - ign);  
  
 }  
 }  
 }  
 }  
  
 }  
  
 **private double**[] rotatedCoords(**double** pointX, **double** pointY,  
 **double** originX, **double** originY, **double** angle) {  
 **var** x = Math.*cos*(Math.*toRadians*(angle)) \* (pointX - originX)  
 + Math.*sin*(Math.*toRadians*(angle)) \* (pointY - originY) + originX;  
 **var** y = -Math.*sin*(Math.*toRadians*(angle)) \* (pointX - originX)  
 + Math.*cos*(Math.*toRadians*(angle)) \* (pointY - originY) + originY;  
 **return new double**[]{x, y};  
  
 }  
  
 **public void** becomeIgnited() {  
 **ignitedByUrban** = **true**;  
 }  
  
 **public void** makeIgnitedByUrbanDefault() {  
 **ignitedByUrban** = **false**;  
 }  
}

* + 1. **Класс ForestStates**

**package** com.model.forest;  
  
**public enum** ForestStates {  
 ***UNBURNED***(0),  
 ***IGNITED***(1),  
 ***DEVELOPING***(2),  
 ***EXTINGUISHING***(3),  
 ***BURNED***(4);  
  
 ForestStates(**int** value) {  
 **this**.**value** = value;  
 }  
  
 **private final int value**;  
  
 **public int** getValue() {  
 **return value**;  
 }  
}

* 1. **Пакет input**
     1. **Класс InputData**

**package** com.model.input;  
  
**import** org.gdal.ogr.Geometry;  
**import** org.gdal.ogr.ogrConstants;  
  
**import** java.time.LocalDateTime;  
  
  
**public class** InputData {  
  
 **int side**;  
  
 **public** String getFuel() {  
 **return fuel**;  
 }  
  
 **public** String getFuelCodes() {  
 **return fuelCodes**;  
 }  
  
 **public** String getElevation() {  
 **return elevation**;  
 }  
  
 **public int** getWeatherPeriod() {  
 **return weatherPeriod**;  
 }  
  
 **public** String getIgnition() {  
 **return ignition**;  
 }  
  
 String **ignition**;  
  
 String **fuel**;  
 String **fuelCodes**;  
 String **elevation**;  
  
 **public** String getBuildingsPath() {  
 **return buildingsPath**;  
 }  
  
 String **buildingsPath**;  
 **int weatherPeriod**;  
  
 **public double** getHouseMaterial() {  
 **return houseMaterial**;  
 }  
  
 **double houseMaterial**;  
  
 **public** InputData(**double**[] coords, String fuel, String fuelCodes, String elevation, String meteodata, **int** meteoDataChange,  
 LocalDateTime start, LocalDateTime finish, **int** side, String ignition, String buildingsPath, **double** houseMaterial) {  
 **this**.**meteodata** = meteodata;  
 **this**.**start** = start;  
 **this**.**finish** = finish;  
 **this**.**fuel** = fuel;  
 **this**.**fuelCodes** = fuelCodes;  
 **this**.**elevation** = elevation;  
 **this**.**weatherPeriod** = meteoDataChange;  
 **this**.**buildingsPath** = buildingsPath;  
 **this**.**houseMaterial** = houseMaterial;  
  
 **startPoint** = **new** Geometry(ogrConstants.***wkbPoint***);  
 **endPoint** = **new** Geometry(ogrConstants.***wkbPoint***);  
 **startPoint**.AddPoint(coords[0], coords[1]);  
 **endPoint**.AddPoint(coords[2], coords[3]);  
  
 **this**.**ignition** = ignition;  
 **this**.**side** = side;  
 }  
  
  
 **public** Geometry getStartPoint() {  
 **return startPoint**;  
 }  
  
 *// coordinates* **private** Geometry **startPoint**;  
  
 **public** Geometry getEndPoint() {  
 **return endPoint**;  
 }  
  
 **private** Geometry **endPoint**;  
  
 **public** String getMeteodata() {  
 **return meteodata**;  
 }  
  
 **private final** String **meteodata**; *// files per time* **public int** getSide() {  
 **return side**;  
 }  
  
 **public** LocalDateTime getStart() {  
 **return start**;  
 }  
  
 **public** LocalDateTime getFinish() {  
 **return finish**;  
 }  
  
 *// start and end of simulation* **private final** LocalDateTime **start**;  
 **private final** LocalDateTime **finish**;  
  
  
}

* 1. **Пакет urban**
     1. **Класс UrbanArea**

**package** com.model.urban;  
  
**import** com.model.forest.ForestCell;  
**import** com.model.input.InputData;  
**import** org.gdal.gdal.Band;  
**import** org.gdal.gdal.Dataset;  
**import** org.gdal.gdal.WarpOptions;  
**import** org.gdal.gdal.gdal;  
**import** org.gdal.gdalconst.gdalconst;  
**import** org.gdal.ogr.\*;  
**import** org.gdal.osr.CoordinateTransformation;  
**import** org.gdal.osr.SpatialReference;  
  
**import** java.io.File;  
**import** java.util.\*;  
  
  
**public class** UrbanArea {  
 **int width**, **length**;  
 **int side**;  
 InputData **inputData**;  
 SpatialReference **spatialReferenceUTM**;  
 String **areaVectorPath** = **"..\\data\\buildings\\urban\_area.shp"**;  
 String **areaRasterPath** = **"..\\data\\buildings\\buildings.tif"**;  
 UrbanCell[][] **urbanCells**;  
 UrbanStates[][] **states**;  
 Random **random** = **new** Random();  
  
 **public** UrbanCell[][] getUrbanCells() {  
 **return urbanCells**;  
 }  
**public** UrbanArea(InputData inputData, SpatialReference spatialReferenceUTM, **int** length, **int** width) {  
 **this**.**width** = width;  
 **this**.**length** = length;  
 **this**.**inputData** = inputData;  
 **this**.**spatialReferenceUTM** = spatialReferenceUTM;  
 **urbanCells** = **new** UrbanCell[width][length];  
 **states** = **new** UrbanStates[width][length];  
 **this**.**side** = inputData.getSide();  
 UrbanCell.*material* = inputData.getHouseMaterial();  
  
 **if** (inputData.getIgnition().endsWith(**"osm"**))  
 extractBuildings(inputData, spatialReferenceUTM);  
 rasterizeBuildingMap();  
 initUrbanCells();  
  
 }  
  
 **private void** initUrbanCells() {  
 **var** dataset = gdal.*Open*(**areaRasterPath**);  
 **var** paths = generatePaths(**areaRasterPath**, **"urban.tif"**);  
 dataset = changeProjection(dataset, paths[0]);  
 dataset = changeResolutionAndBorders(dataset, paths[1]);  
  
 **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** transform = **new** CoordinateTransformation(sourceSRS, **spatialReferenceUTM**);  
  
 **double**[] start = transform.TransformPoint(**inputData**.getStartPoint().GetX(),  
 **inputData**.getStartPoint().GetY());  
  
 **double** x, y;  
  
 **var** band = dataset.GetRasterBand(1);  
 **int**[] presence = **new int**[1];  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 band.ReadRaster(i, **length** - 1 - j, 1, 1, presence);  
 x = start[1] + i \* **side**;  
 y = start[0] + j \* **side**;  
  
 **if** (presence[0] > 0) {  
 **urbanCells**[i][j] = **new** UrbanCell(x, y, **side**);  
 **states**[i][j] = UrbanStates.***UNBURNED***;  
 }  
  
 }  
 }  
  
 band.delete();  
 dataset.delete();  
 }  
  
 **private void** rasterizeBuildingMap() {  
 **var** urbanData = ogr.*Open*(**areaVectorPath**);  
 **var** urbanLayer = urbanData.GetLayer(0);  
  
 SpatialReference sourceSrs = urbanLayer.GetSpatialRef();  
 **double**[] extent = urbanLayer.GetExtent();  
  
 **double** x\_res = ((extent[1] - extent[0]) / **side**);  
 **double** y\_res = ((extent[3] - extent[2]) / **side**);  
  
 **int** xCor = (**int**) x\_res;  
 **int** yCor = (**int**) y\_res;  
  
 Dataset target\_ds = gdal.*GetDriverByName*(**"GTiff"**)  
 .Create(**areaRasterPath**, xCor, yCor, 1, gdalconst.***GDT\_Byte***);  
 target\_ds.SetProjection(sourceSrs.ExportToPrettyWkt());  
 target\_ds.SetGeoTransform(**new double**[]{extent[0], **side**, 0, extent[3], 0, -**side**});  
 Band band = target\_ds.GetRasterBand(1);  
  
  
 **int**[] intArr = {1};  
  
 *// Rasterize* gdal.*RasterizeLayer*(target\_ds, intArr, urbanLayer, **null**);  
  
 urbanLayer.delete();  
 urbanData.delete();  
 target\_ds.delete();  
 band.delete();  
 }  
  
 */\* public void initIgnition(String path) {  
  
 }\*/* **public void** propagate(**double** step) {  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **if** (**urbanCells**[i][j] == **null**)  
 **continue**;  
  
 **switch** (**urbanCells**[i][j].getState()) {  
 **case *IGNITED*** -> {  
 **urbanCells**[i][j].developIgnition(step, **states**, **random**, i, j);  
 }  
 **case *SLOWDEVELOPING***, ***FULLDEVELOPMENT*** -> {  
 **double** a = 3 \* **urbanCells**[i][j].getWindVelocity() / 5 + 3 + **side** / 2.0;  
 **double** b = -2 \* **urbanCells**[i][j].getWindVelocity() / 15 + 3 + **side** / 2.0;  
 **double** c = -1 \* **urbanCells**[i][j].getWindVelocity() / 15 + 3 + **side** / 2.0;  
 **var** t = Math.*sqrt*(b \* (a + c) / 2.0);  
 **var** geom = Geometry.*CreateFromWkt*(**urbanCells**[i][j].getGeometry()).Centroid();  
 **double** x = geom.GetX(), y = geom.GetY();  
 **var** influence = **new** Geometry(ogr.***wkbLinearRing***);  
 **double**[] f = rotatedCoords(x - t, y + c, x, y, **urbanCells**[i][j].getWindAngle());  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x + t, y + c, x, y, **urbanCells**[i][j].getWindAngle());  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x + t, y - a, x, y, **urbanCells**[i][j].getWindAngle());  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x - t, y - a, x, y, **urbanCells**[i][j].getWindAngle());  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x - t, y + c, x, y, **urbanCells**[i][j].getWindAngle());  
 influence.AddPoint(f[0], f[1]);  
  
 **var** influenceArea = **new** Geometry(ogr.***wkbPolygon***);  
 influenceArea.AddGeometry(influence);  
  
  
 **int** mini = (**int**) Math.*max*(0, i - a / **side**);  
 **int** minj = (**int**)Math.*max*(0, j - a / **side**);  
 **int** maxi = (**int**) Math.*min*(**width**, i + a / **side**);  
 **int** maxj = (**int**) Math.*min*(**length**, j + a / **side**);  
  
  
 **double** ign;  
 **for** (**int** l = mini; l < maxi; l++) {  
 **for** (**int** m = minj; m < maxj; m++) {  
 **if** (**urbanCells**[l][m] != **null** && **urbanCells**[l][m].getState().equals(UrbanStates.***UNBURNED***)){  
 **var** urbanGeom = Geometry.*CreateFromWkt*(**urbanCells**[l][m].getGeometry());  
 **if** (urbanGeom.Intersection(influenceArea) != **null**){  
 ign = **urbanCells**[l][m].getMaterial() \* **urbanCells**[l][m].getWeather()  
 \* urbanGeom.Intersection(influenceArea).Area() / urbanGeom.Area();  
  
 **if** (**urbanCells**[i][j].getState().equals(UrbanStates.***SLOWDEVELOPING***))  
 ign \*= 0.4;  
 **urbanCells**[l][m].addIgnitionProbability(1 - ign);  
 }  
 }  
 }  
 }  
  
 **urbanCells**[i][j]  
 .fireSpreadOnUrban(step, **states**, **random**, i, j);  
 }  
  
 **case *FLASHOVER*** -> **urbanCells**[i][j].developFlashover(step, **states**, **random**, i, j);  
 **case *UNBURNED***, ***EXTINGUISHED*** -> {  
 }  
 }  
 }  
 }  
 }  
  
 **private double**[] rotatedCoords(**double** pointX, **double** pointY,  
 **double** originX, **double** originY, **double** angle) {  
 **var** x = Math.*cos*(Math.*toRadians*(angle)) \* (pointX - originX)  
 + Math.*sin*(Math.*toRadians*(angle)) \* (pointY - originY) + originX;  
 **var** y = -Math.*sin*(Math.*toRadians*(angle)) \* (pointX - originX)  
 + Math.*cos*(Math.*toRadians*(angle)) \* (pointY - originY) + originY;  
 **return new double**[]{x, y};  
  
 }  
  
  
  
 **private void** extractBuildings(InputData inputData, SpatialReference spatialReferenceUTM) {  
 gdal.*AllRegister*();  
 **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** transform = **new** CoordinateTransformation(sourceSRS, spatialReferenceUTM);  
 **double**[] start = transform.TransformPoint(inputData.getStartPoint().GetX(),  
 inputData.getStartPoint().GetY());  
  
 **double**[] point;  
 **var** data = ogr.*Open*(inputData.getBuildingsPath());  
 **var** layer = data.GetLayerByName(**"multipolygons"**);  
 **var** source = layer.GetSpatialRef();  
  
 **var** trans = **new** CoordinateTransformation(source, spatialReferenceUTM);  
  
 **var** driver = gdal.*GetDriverByName*(**"ESRI Shapefile"**);  
 **var** dataset = driver.Create(**areaVectorPath**, 0, 0,  
 1, gdalconst.***GDT\_Unknown***, (String[]) **null**);  
 **var** dataLayer = dataset.CreateLayer(**"houses"**,  
 spatialReferenceUTM, ogrConstants.***wkbPolygon***);  
  
 **var** id = **new** FieldDefn(**"id"**, ogr.***OFTInteger***);  
 dataLayer.CreateField(id);  
  
 Feature f;  
 **while** ((f = layer.GetNextFeature()) != **null**) {  
 **for** (**int** i = 0; i < f.GetFieldCount(); i++) {  
 **if** (**"house"**.equals(f.GetFieldAsString(**"building"**))) {  
 **var** feature = **new** Feature(dataLayer.GetLayerDefn());  
  
 **var** geom = f.GetGeometryRef();  
 geom.TransformTo(spatialReferenceUTM);  
  
 feature.SetGeometry(f.GetGeometryRef());  
 feature.SetField(**"id"**, f.GetFID());  
  
 dataLayer.CreateFeature(feature);  
 feature.delete();  
 }  
 }  
 }  
  
 layer.delete();  
 data.delete();  
 dataset.delete();  
 dataLayer.delete();  
  
 }  
  
  
 **private** String[] generatePaths(String path, String name) {  
 **var** ind = path.lastIndexOf(File.***separator***);  
 **var** projectedPath = path.substring(0, ind + 1) + **"projected\_"** + name;  
 **var** modifiedPath = path.substring(0, ind + 1) + **"modified\_"** + name;  
 **return new** String[]{projectedPath, modifiedPath};  
  
 }  
  
 **private** Dataset changeResolutionAndBorders(Dataset dataset, String path) {  
 *// Изменить размер и разрешение* **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** targetSRS = dataset.GetSpatialRef();  
  
 **var** ct = **new** CoordinateTransformation(sourceSRS, targetSRS);  
  
 **var** beginning = ct.TransformPoint(**inputData**.getStartPoint().GetX(), **inputData**.getStartPoint().GetY());  
 **var** finish = ct.TransformPoint(**inputData**.getEndPoint().GetX(), **inputData**.getEndPoint().GetY());  
  
 Vector<String> options =  
 **new** Vector<>(Arrays.*asList*(**"-te"**, String.*valueOf*(beginning[0]), String.*valueOf*(beginning[1]),  
 String.*valueOf*(finish[0]), String.*valueOf*(finish[1]),  
 **"-tr"**, String.*valueOf*(**inputData**.getSide()), String.*valueOf*(**inputData**.getSide())));  
  
 **var** warpOptions = **new** WarpOptions(options);  
 Dataset[] srcData = {dataset};  
 Dataset modified = gdal.*Warp*(path, srcData, warpOptions);  
 **return** modified;  
 }  
  
 **private** Dataset changeProjection(Dataset dataset, String path) {  
 Vector<String> options = **new** Vector<>();  
 options.add(**"-t\_srs"**);  
 options.add(**spatialReferenceUTM**.ExportToPrettyWkt());  
 WarpOptions warpOptions = **new** WarpOptions(options);  
 Dataset[] srcData = {dataset};  
 Dataset projected = gdal.*Warp*(path, srcData, warpOptions);  
 dataset = gdal.*Open*(path);  
  
 **return** dataset;  
 }  
  
  
 **public void** propagateInForest(ForestCell[][] cells) {  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **if** (**urbanCells**[i][j] == **null**)  
 **continue**;  
  
 **if** (**urbanCells**[i][j].getState().equals(UrbanStates.***SLOWDEVELOPING***) ||  
 **urbanCells**[i][j].getState().equals(UrbanStates.***FULLDEVELOPMENT***))  
 **urbanCells**[i][j].fireSpreadOnForest(cells, i, j, **width**, **length**);  
 }  
 }  
  
 }  
  
 **public void** updateStates() {  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **if** (**urbanCells**[i][j] != **null**) {  
 **urbanCells**[i][j].setState(**states**[i][j]);  
  
 **if** (**urbanCells**[i][j].getState().equals(UrbanStates.***UNBURNED***)) {  
 **if** (**urbanCells**[i][j].getIgnitionProbability() > 0) {  
  
 **if** (**random**.nextDouble() <= (1 - **urbanCells**[i][j].getIgnitionProbability())) {  
 **urbanCells**[i][j].setState(UrbanStates.***IGNITED***);  
 System.***out***.println(**"Ignited!"**);  
 **states**[i][j] = UrbanStates.***IGNITED***;  
 }  
 **urbanCells**[i][j].setIgnitionProbability(1.0);  
 }  
 }  
 }  
 }  
 }  
  
 }  
  
 **public void** printUrbanStatistics() {  
 **int** ignited = 0, unburned = 0, sldevelop = 0, fulldevelop = 0,  
 exting = 0, flash = 0;  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **if** (**states**[i][j] == **null**)  
 **continue**;  
 **switch**(**states**[i][j]){  
 **case *UNBURNED*** -> {  
 unburned++;  
 }  
 **case *IGNITED*** -> {  
 ignited++;  
 }  
 **case *SLOWDEVELOPING*** -> {  
 sldevelop++;  
 }  
 **case *FULLDEVELOPMENT*** -> {  
 fulldevelop++;  
 }  
 **case *FLASHOVER*** -> {  
 flash++;  
 }  
 **case *EXTINGUISHED*** -> {  
 exting++;  
 }  
 }  
 }  
 }  
  
 System.***out***.println(**"++++URBAN CELLS+++++"**);  
 System.***out***.println(**"UNBURNED = "** + unburned);  
 System.***out***.println(**"IGNITED = "** + ignited);  
 System.***out***.println(**"SLOW DEVELOP = "** + sldevelop);  
 System.***out***.println(**"FULL DEVELOP = "** + fulldevelop);  
 System.***out***.println(**"FLASHOVER = "** + flash);  
 System.***out***.println(**"EXTINGUISHED = "** + exting);  
 }  
  
 **public void** setWeatherData(String weatherDataPath) {  
 **var** dataset = gdal.*Open*(weatherDataPath);  
 Band velocity = dataset.GetRasterBand(1);  
 Band angle = dataset.GetRasterBand(2);  
 Band humidity = dataset.GetRasterBand(4);  
  
 **double** h;  
 **var** hum = **new double**[1];  
 **var** vel = **new double**[1];  
 **var** ang = **new double**[1];  
  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 **if** (**urbanCells**[i][j] != **null**) {  
 humidity.ReadRaster(i, j, 1, 1, hum);  
 velocity.ReadRaster(i, j, 1, 1, vel);  
 angle.ReadRaster(i, j, 1, 1, ang);  
  
 **urbanCells**[i][j].setWindAngle(ang[0]);  
 **urbanCells**[i][j].setWindVelocity(vel[0]);  
  
 **if** (hum[0] < 30) {  
 h = 1.0;  
 } **else if** (hum[0] < 60) {  
 h = 0.8;  
 } **else** h = 0.4;  
  
 **if** (**urbanCells**[i][j] != **null**)  
 **urbanCells**[i][j].setWeather(h);  
  
 }  
 }  
 }  
  
 humidity.delete();  
 velocity.delete();  
 angle.delete();  
 dataset.delete();  
 }  
}

* + 1. **Класс UrbanCell**

**package** com.model.urban;  
  
**import** com.model.forest.ForestCell;  
**import** com.model.forest.ForestStates;  
**import** org.gdal.ogr.Geometry;  
**import** org.gdal.ogr.ogr;  
  
**import** java.util.\*;  
  
**public class** UrbanCell {  
 **double weather**;  
 String **geometry**;  
 **double windVelocity**;  
 **double windAngle**;  
 UrbanStates **state**;  
 **double side**;  
  
 **public double** getWeather() {  
 **return weather**;  
 }  
  
 **public** String getGeometry() {  
 **return geometry**;  
 }  
  
 **public** UrbanCell(**double** x, **double** y, **double** side) {  
 **this**.**state** = UrbanStates.***UNBURNED***;  
 **this**.**side** = side;  
 Geometry poly = calculateGeometry(x, y);  
 **this**.**geometry** = poly.ExportToWkt();  
  
 }  
  
 **private** Geometry calculateGeometry(**double** x, **double** y) {  
 **var** ring = **new** Geometry(ogr.***wkbLinearRing***);  
 ring.AddPoint(x, y);  
 ring.AddPoint(x + **side**, y);  
 ring.AddPoint(x + **side**, y + **side**);  
 ring.AddPoint(x, y + **side**);  
 ring.AddPoint(x, y);  
  
 **var** poly = **new** Geometry(ogr.***wkbPolygon***);  
 poly.AddGeometry(ring);  
  
 **return** poly;  
 }  
  
 **public double** getWindVelocity() {  
 **return windVelocity**;  
 }  
  
 **public double** getSide() {  
 **return side**;  
 }  
  
  
 **public double** getIgnitionProbability() {  
 **return ignitionProbability**;  
 }  
  
 **public void** setIgnitionProbability(**double** ignitionProbability) {  
 **this**.**ignitionProbability** = ignitionProbability;  
 }  
  
 **double ignitionProbability** = 1.0;  
  
  
 **public double** getWindAngle() {  
 **return windAngle**;  
 }  
  
 **public** UrbanStates getState() {  
 **return state**;  
 }  
  
 **double**[] **coords**;  
  
 **public void** setState(UrbanStates state) {  
 **this**.**state** = state;  
 }  
  
 **public double** getMaterial() {  
 **return** *material*;  
 }  
  
 **public double** getInnerTime() {  
 **return innerTime**;  
 }  
  
 **public void** setInnerTime(**double** innerTime) {  
 **this**.**innerTime** = innerTime;  
 }  
  
 **double innerTime** = 0;  
  
 **public void** setMaterial(**double** material) {  
 **this**.*material* = material;  
 }  
  
 **public void** setWeather(**double** weather) {  
 **this**.**weather** = weather;  
 }  
  
 **public void** setWindVelocity(**double** windVelocity) {  
 **this**.**windVelocity** = windVelocity;  
 }  
  
 **public void** setWindAngle(**double** windAngle) {  
 **this**.**windAngle** = windAngle;  
 }  
  
  
 **public** Geometry calculateAreaOfInterest() {  
 **var** a = 3 \* **windVelocity** / 5 + 3 + **side** / 2;  
 **var** pt = Geometry.*CreateFromWkt*(**geometry**).Centroid();  
 **return** pt.Buffer(a);  
 }  
  
 **public void** fireSpreadOnUrban(**double** step,  
 UrbanStates[][] states, Random rand, **int** i, **int** j) {  
  
 **if** (**innerTime** == 0) {  
 **if** (**state**.equals(UrbanStates.***SLOWDEVELOPING***))  
 **innerTime** = (rand.nextDouble() \* 3 + 5) \* 60;  
 **else** {  
 **if** (*material* == 1.0) {  
 **innerTime** = (rand.nextDouble() \* 10 + 10) \* 60;  
 } **else if** (*material* == 0.8)  
 **innerTime** = (rand.nextDouble() \* 10 + 20) \* 60;  
 **else  
 innerTime** = (rand.nextDouble() \* 10 + 30) \* 60;  
 }  
 } **else** {  
 **innerTime** -= step;  
 **if** (**innerTime** <= 0) {  
 **innerTime** = 0;  
 states[i][j] = UrbanStates.*values*()[**state**.getValue() + 1];  
 }  
 }  
  
 }  
  
 **public void** developIgnition(**double** step, UrbanStates[][] states, Random rand, **int** i, **int** j) {  
 **if** (**innerTime** == 0) {  
 **innerTime** = (rand.nextDouble() \* 2 + 4) \* 60;  
 } **else** {  
 **innerTime** -= step;  
 **if** (**innerTime** <= 0) {  
 **innerTime** = 0;  
 states[i][j] = UrbanStates.***SLOWDEVELOPING***;  
 }  
 }  
 }  
  
 **public void** developFlashover(**double** step, UrbanStates[][] states, Random rand, **int** i, **int** j) {  
 **if** (**innerTime** == 0) {  
 **if** (*material* == 1.0) {  
 **innerTime** = (rand.nextDouble() \* 10 + 20) \* 60;  
 } **else if** (*material* == 0.8)  
 **innerTime** = (rand.nextDouble() \* 10 + 30) \* 60;  
 **else  
 innerTime** = (rand.nextDouble() \* 10 + 50) \* 60;  
 } **else** {  
 **innerTime** -= step;  
 **if** (**innerTime** <= 0) {  
 **innerTime** = 0;  
 states[i][j] = UrbanStates.***EXTINGUISHED***;  
 }  
 }  
 }  
  
 **static double** *material* = 1.0;  
  
 **public void** fireSpreadOnForest(ForestCell[][] cells, **int** i, **int** j, **int** width, **int** length) {  
 **double** a = 3 \* getWindVelocity() / 5 + 3 + **side** / 2;  
 **double** b = -2 \* getWindVelocity() / 15 + 3 + **side** / 2;  
 **double** c = -1 \* getWindVelocity() / 15 + 3 + **side** / 2;  
  
  
 **var** t = Math.*sqrt*(b \* (a + c) / 2.0);  
 **var** geom = Geometry.*CreateFromWkt*(**geometry**).Centroid();  
 **double** x = geom.GetX(), y = geom.GetY();  
 **var** influence = **new** Geometry(ogr.***wkbLinearRing***);  
 **double**[] f = rotatedCoords(x - t, y + c, x, y, **windAngle**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x + t, y + c, x, y, **windAngle**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x + t, y - a, x, y, **windAngle**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x - t, y - a, x, y, **windAngle**);  
 influence.AddPoint(f[0], f[1]);  
 f = rotatedCoords(x - t, y + c, x, y, **windAngle**);  
 influence.AddPoint(f[0], f[1]);  
  
 **var** influenceArea = **new** Geometry(ogr.***wkbPolygon***);  
 influenceArea.AddGeometry(influence);  
  
  
 **int** mini = (**int**) Math.*max*(0, i - a / **side**);  
 **int** minj = (**int**) Math.*max*(0, j - a / **side**);  
 **int** maxi = (**int**) Math.*min*(width, i + a / **side**);  
 **int** maxj = (**int**) Math.*min*(length, j + a / **side**);  
  
 **for** (**int** l = mini; l < maxi; l++) {  
 **for** (**int** m = minj; m < maxj; m++) {  
 **if** (cells[l][m].getState() != ForestStates.***UNBURNED***)  
 **continue**;  
 **var** forestGeom = Geometry.*CreateFromWkt*(cells[l][m].getGeometry());  
 **if** (influenceArea.Intersect(forestGeom))  
 cells[l][m].becomeIgnited();  
 }  
 }  
  
 }  
  
 **private double**[] rotatedCoords(**double** pointX, **double** pointY,  
 **double** originX, **double** originY, **double** angle) {  
 **var** x = Math.*cos*(Math.*toRadians*(angle)) \* (pointX - originX)  
 + Math.*sin*(Math.*toRadians*(angle)) \* (pointY - originY) + originX;  
 **var** y = -Math.*sin*(Math.*toRadians*(angle)) \* (pointX - originX)  
 + Math.*cos*(Math.*toRadians*(angle)) \* (pointY - originY) + originY;  
 **return new double**[]{x, y};  
  
 }  
  
 **public void** addIgnitionProbability(**double** v) {  
 **ignitionProbability** \*= v;  
 }

* + 1. **Класс UrbanStates**

**package** com.model.urban;  
  
**public enum** UrbanStates {  
 ***UNBURNED***(0),  
 ***IGNITED***(1),  
 ***SLOWDEVELOPING***(2),  
 ***FULLDEVELOPMENT***(3),  
 ***FLASHOVER***(4),  
 ***EXTINGUISHED***(5);  
  
 UrbanStates(**int** value) {  
 **this**.**value** = value;  
 }  
  
 **private final int value**;  
  
 **public int** getValue() {  
 **return value**;  
 }  
}

* 1. **Класс GlobalFire**

**package** com.model;  
  
**import** com.opencsv.CSVReader;  
**import** com.opencsv.exceptions.CsvValidationException;  
**import** com.model.forest.ForestArea;  
**import** com.model.input.InputData;  
**import** org.gdal.gdal.Band;  
**import** org.gdal.gdal.Dataset;  
**import** org.gdal.gdal.WarpOptions;  
**import** org.gdal.gdal.gdal;  
**import** org.gdal.gdalconst.gdalconst;  
**import** org.gdal.osr.CoordinateTransformation;  
**import** org.gdal.osr.SpatialReference;  
**import** com.model.urban.UrbanArea;  
  
**import** java.io.File;  
**import** java.io.FileReader;  
**import** java.io.IOException;  
**import** java.time.Duration;  
**import** java.time.LocalDateTime;  
**import** java.time.format.DateTimeFormatter;  
**import** java.util.Arrays;  
**import** java.util.Vector;  
  
**public class** GlobalFire {  
 InputData **inputData**;  
 ForestArea **forest**;  
 UrbanArea **urban**;  
 SpatialReference **spatialReferenceUTM**;  
 **int side**;  
 **int length**;  
  
 **public** SpatialReference getSpatialReferenceUTM() {  
 **return spatialReferenceUTM**;  
 }  
  
 **public int** getLength() {  
 **return length**;  
 }  
  
 **public int** getWidth() {  
 **return width**;  
 }  
  
 **public** LocalDateTime getCurrentDate() {  
 **return currentDate**;  
 }  
  
 **int width**;  
 LocalDateTime **currentDate**;  
  
 **public** GlobalFire(InputData inputData) {  
 **this**.**inputData** = inputData;  
 **this**.**side** = inputData.getSide();  
  
 initSpatialReference();  
 defineAreaSize();  
  
 **currentDate** = inputData.getStart();  
 **forest** = **new** ForestArea(inputData, **spatialReferenceUTM**, **length**, **width**);  
 **urban** = **new** UrbanArea(inputData, **spatialReferenceUTM**, **length**, **width**);  
 setWeather(inputData.getMeteodata(), 0);  
  
 }  
  
 **private void** setWeather(String weather, **int** number) {  
 FileReader file = **null**;  
 **try** {  
 file = **new** FileReader(weather);  
  
 **var** csvReader = **new** CSVReader(file);  
 String[] record;  
 csvReader.skip(number);  
 **int** ind = weather.lastIndexOf(**"/"**);  
 **if** (ind == -1)  
 ind = weather.lastIndexOf(**"\\"**);  
  
 String dir = weather.substring(0, ind + 1);  
  
 record = csvReader.readNext();  
  
 String weatherPath = mergeWeatherData(dir, record);  
 **forest**.setWeatherData(weatherPath);  
 **urban**.setWeatherData(weatherPath);  
  
 csvReader.close();  
 }  
 **catch** (CsvValidationException | IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private** String mergeWeatherData(String dir, String[] record) {  
 String output = dir + **"weather"** + **currentDate** .format(DateTimeFormatter.*ofPattern*(**"yyyy\_MM\_dd\_HH\_mm"**)) + **".tif"**;  
 **var** dataset = gdal.*GetDriverByName*(**"GTiff"**).Create(output,  
 **width**, **length**, 4, gdalconst.***GDT\_Float64***);  
 dataset.SetProjection(**spatialReferenceUTM**.ExportToPrettyWkt());  
 **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** transform = **new** CoordinateTransformation(sourceSRS, **spatialReferenceUTM**);  
  
 **double**[] start = transform.TransformPoint(**inputData**.getStartPoint().GetX(),  
 **inputData**.getStartPoint().GetY());  
  
 dataset.SetGeoTransform(**new double**[]{start[0], **side**, 0, start[1], 0, -**side**});  
  
 String projectedName = **"wind\_"** + **currentDate** .format(DateTimeFormatter.*ofPattern*(**"yyyy\_MM\_dd\_HH\_mm"**)) + **"\_vel.tif"**;  
 addBandToWeatherDataset(dir + record[1], dataset, projectedName, 1);  
  
 projectedName = **"wind\_"** + **currentDate** .format(DateTimeFormatter.*ofPattern*(**"yyyy\_MM\_dd\_HH\_mm"**)) + **"\_ang.tif"**;  
 addBandToWeatherDataset(dir + record[2], dataset, projectedName, 2);  
  
 projectedName = **"temp\_"** + **currentDate** .format(DateTimeFormatter.*ofPattern*(**"yyyy\_MM\_dd\_HH\_mm"**)) + **".tif"**;  
 addBandToWeatherDataset(dir + record[3], dataset, projectedName, 3);  
  
 projectedName = **"hum\_"** + **currentDate** .format(DateTimeFormatter.*ofPattern*(**"yyyy\_MM\_dd\_HH\_mm"**)) + **".tif"**;  
 addBandToWeatherDataset(dir + record[4], dataset, projectedName, 4);  
  
  
 dataset.delete();  
 **return** output;  
 }  
  
 **private void** addBandToWeatherDataset(String name, Dataset dataset, String projectedName, **int** bandNumber) {  
 Dataset originalDataset = gdal.*Open*(name);  
  
 **var** paths = generatePaths(name, projectedName);  
  
 originalDataset = changeProjection(originalDataset, paths[0]);  
 originalDataset = changeResolutionAndBorders(originalDataset, paths[1]);  
  
 Band original = originalDataset.GetRasterBand(1);  
 Band band = dataset.GetRasterBand(bandNumber);  
  
 **var** data = **new double**[**width**];  
 **for** (**int** i = **length** - 1; i >= 0; i--) {  
 original.ReadRaster(0, i, data.**length**, 1, data);  
 band.WriteRaster(0, i, data.**length**, 1, data);  
 }  
  
 original.delete();  
 originalDataset.delete();  
 }  
  
 **private** Dataset changeResolutionAndBorders(Dataset dataset, String path) {  
 *// Изменить размер и разрешение* **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** targetSRS = dataset.GetSpatialRef();  
  
 **var** ct = **new** CoordinateTransformation(sourceSRS, targetSRS);  
  
 **var** beginning = ct.TransformPoint(**inputData**.getStartPoint().GetX(),  
 **inputData**.getStartPoint().GetY());  
 **var** finish = ct.TransformPoint(**inputData**.getEndPoint().GetX(),  
 **inputData**.getEndPoint().GetY());  
  
 Vector<String> options =  
 **new** Vector<>(Arrays.*asList*(**"-te"**, String.*valueOf*(beginning[0]),  
 String.*valueOf*(beginning[1]),  
 String.*valueOf*(finish[0]), String.*valueOf*(finish[1]),  
 **"-tr"**, String.*valueOf*(**side**), String.*valueOf*(**side**)));  
  
 **var** warpOptions = **new** WarpOptions(options);  
 Dataset[] srcData = {dataset};  
 Dataset modified = gdal.*Warp*(path, srcData, warpOptions);  
 **return** modified;  
 }  
  
 **private** Dataset changeProjection(Dataset dataset, String path) {  
 Vector<String> options = **new** Vector<>();  
 options.add(**"-t\_srs"**);  
 options.add(**spatialReferenceUTM**.ExportToPrettyWkt());  
 WarpOptions warpOptions = **new** WarpOptions(options);  
 Dataset[] srcData = {dataset};  
 Dataset projected = gdal.*Warp*(path, srcData, warpOptions);  
 dataset = gdal.*Open*(path);  
  
 **return** dataset;  
 }  
  
 **private** String[] generatePaths(String path, String name) {  
 **var** ind = path.lastIndexOf(File.***separator***);  
 **var** projectedPath = path.substring(0, ind + 1) + **"projected\_"** + name;  
 **var** modifiedPath = path.substring(0, ind + 1) + **"modified\_"** + name;  
 **return new** String[]{projectedPath, modifiedPath};  
  
 }  
  
 **public void** propagate() {  
 **int** step = 90;  
 **double** minutesLeft = 0;  
 System.***out***.println(**currentDate**);  
  
 **forest**.printStatistics();  
 **urban**.printUrbanStatistics();  
  
 **while** (**currentDate**.compareTo(**inputData**.getFinish()) < 0) {  
 **forest**.propagate(minutesLeft, step, **currentDate**);  
 **forest**.propagateInUrban(**urban**.getUrbanCells());  
 **urban**.propagate(step);  
 **urban**.propagateInForest(**forest**.getCells());  
  
 **forest**.updateStates();  
 **urban**.updateStates();  
  
 **currentDate** = **currentDate**.plusSeconds(step);  
 minutesLeft += (step / 60);  
 **if** (minutesLeft == **inputData**.getWeatherPeriod()) {  
 setWeather(**inputData**.getMeteodata(),  
 (**int**) Duration.*between*(**inputData**.getStart(), **currentDate**).toHours());  
  
 **forest**.printStatistics();  
 **urban**.printUrbanStatistics();  
 presentResult();  
 minutesLeft = 0;  
 }  
 }  
 presentResult();  
  
 }  
  
 **private void** presentResult() {  
 presentForestResults();  
 }  
  
 **private void** presentForestResults() {  
 String path = **"..\\data\\result\\result\_"** + **currentDate** .format(DateTimeFormatter.*ofPattern*(**"yyyy\_MM\_dd\_HH\_mm"**)) + **".tif"**;  
  
 Dataset resultData = gdal.*GetDriverByName*(**"GTiff"**).Create(path,  
 **width**, **length**,  
 1, gdalconst.***GDT\_Byte***);  
  
  
 **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
 **var** ct = **new** CoordinateTransformation(sourceSRS, **spatialReferenceUTM**);  
 **var** beginning = ct.TransformPoint(**inputData**.getStartPoint().GetX(),  
 **inputData**.getStartPoint().GetY());  
  
 **double**[] geotransform = {beginning[0], 30, 0.0, beginning[1], 0, -30};  
 resultData.SetGeoTransform(geotransform);  
 resultData.SetProjection(**spatialReferenceUTM**.ExportToPrettyWkt());  
  
 Band result = resultData.GetRasterBand(1);  
  
 **byte** value = 0;  
 **for** (**int** i = 0; i < **width**; i++) {  
 **for** (**int** j = 0; j < **length**; j++) {  
 result.WriteRaster(i, j, 1, 1,  
 **new byte**[]{(**byte**) **forest**.getCells()[i][j].getState().getValue()});  
 **if** (**urban**.getUrbanCells()[i][j] != **null**)  
 value = (**byte**) **urban**.getUrbanCells()[i][j].getState().getValue();  
 **if** (value > 0)  
 result.WriteRaster(i, j, 1, 1, **new byte**[]{value});  
  
 }  
 }  
  
 result.delete();  
 resultData.delete();  
 }  
  
 **private void** initSpatialReference() {  
 **this**.**spatialReferenceUTM** = **new** SpatialReference();  
 **int** zone = (**int**) Math.*round*(30 + **inputData**.getStartPoint().GetY() / 6);  
 **spatialReferenceUTM**.SetProjCS(String.*format*(**"UTM %d (WGS84)"**, zone));  
 **spatialReferenceUTM**.SetWellKnownGeogCS(**"WGS84"**);  
 **spatialReferenceUTM**.SetUTM(zone);  
  
 }  
  
 **private void** defineAreaSize() {  
 **var** sourceSRS = **new** SpatialReference();  
 sourceSRS.ImportFromEPSG(4326);  
  
 **var** ct = **new** CoordinateTransformation(sourceSRS, **spatialReferenceUTM**);  
 **var** beginning = ct.TransformPoint(**inputData**.getStartPoint().GetX(),  
 **inputData**.getStartPoint().GetY());  
 **var** finish = ct.TransformPoint(**inputData**.getEndPoint().GetX(),  
 **inputData**.getEndPoint().GetY());  
  
  
 **width** = (**int**) Math.*round*(Math.*abs*(beginning[0] - finish[0]) / **side**);  
 **length** = (**int**) Math.*round*(Math.*abs*(beginning[1] - finish[1]) / **side**);  
 }  
}

* 1. **Класс Main**

**package** com.model;  
  
**import** com.model.input.InputData;  
**import** org.gdal.gdal.gdal;  
  
**import** java.time.LocalDateTime;  
**import** java.time.format.DateTimeFormatter;  
  
**public class** Main {  
 **public static void** main(String[] args){  
 gdal.*AllRegister*();  
  
 **int** side = Integer.*parseInt*(args[0]); *//30; // m* String elevation = args[1]; *//"..\\data\\elevation\\US\_DEM2016\\US\_DEM2016.tif";* String fuel = args[2]; *//"..\\data\\US\_200EVT\\US\_200EVT.tif";* String csvfueltypes = args[3];*//"..\\data\\US\_200EVT\\LF16\_EVT\_200.csv";* String weather = args[4]; *//"C:\\Users\\admin\\Documents\\firemodel\\project\\data\\weather\\weather.csv";* String ignition = args[5]; *//"C:\\Users\\admin\\Documents\\firemodel\\project\\data\\ignition\\ignition.shp";* DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*(**"yyyy.MM.dd HH:mm"**);  
 LocalDateTime start = LocalDateTime.*parse*(args[6].concat(**" "** + args[7]), formatter); *//LocalDateTime.of(2019, 10,28, 1, 34);* **var** finish = LocalDateTime.*parse*(args[8].concat(**" "** + args[9]), formatter);;*//LocalDateTime.of(2019, 10, 28, 11, 34); // 11* **int** weatherPeriod = Integer.*parseInt*(args[10]); *//60; // minutes* **double** houseMaterial = Double.*parseDouble*(args[11]); *//1.0;* **double**[] coords = {Double.*parseDouble*(args[12]), Double.*parseDouble*(args[13]),  
 Double.*parseDouble*(args[14]), Double.*parseDouble*(args[15])};*//{34.11, -118.50, 34.07, -118.47};* **var** buildings = args[16]; *//"C:\\Users\\admin\\Documents\\firemodel\\project\\data\\buildings\\map.osm";* **var** input = **new** InputData(coords, fuel, csvfueltypes, elevation, weather, weatherPeriod,  
 start, finish, side, ignition, buildings, houseMaterial);  
  
 **var** globalFire = **new** GlobalFire(input);  
 globalFire.propagate();  
 }  
}

1. **Подсистема визуализации**
   1. **Файл UIWindow.py**

**from** UI.UIElements **import** UIElement, AddLayerWindow, ModelSettingsWindow  
  
WINDOW\_SIZE = 0  
  
  
**class** UIWindows:  
 MAIN\_WINDOW\_OBJECTS = [(QMenuBar, **"menubar"**), (QHBoxLayout, **"mainContent"**), (QListWidget, **"listOfLayers"**)]  
  
 **def** \_\_init\_\_(self):  
 self.main\_window = UIElement(UIWindows.MAIN\_WINDOW\_OBJECTS, **"UI/MainWindow.ui"**, QMainWindow())  
 self.add\_layer\_window = AddLayerWindow(**"UI/AddLayerWindow.ui"**, self.main\_window, self)  
 self.launch\_model\_window = ModelSettingsWindow(**"UI/ModelSettingsWindow.ui"**, self.main\_window, self)  
  
 self.loading\_view = **None** self.web = QWebEngineView()  
 self.main\_window.elements[**"mainContent"**].addWidget(self.web, stretch=1)  
 self.map\_view = MapView(self.web, ui=self)  
  
 self.add\_layer\_window.initialize()  
 self.launch\_model\_window.initialize()  
  
 self.initialize\_menubar()  
  
 self.main\_window.elements[**"listOfLayers"**].setContextMenuPolicy(Qt.CustomContextMenu)  
 self.main\_window.elements[**"listOfLayers"**].customContextMenuRequested.connect(self.show\_layers\_context\_menu)  
 self.main\_window.element.show()  
  
 **def** initialize\_menubar(self):  
 self.main\_window.element.findChild(QAction, **"actionNew\_project"**).triggered.connect(self.new\_project)self.main\_window.element.findChild(QAction, **"actionExit"**) \  
 .triggered.connect(**lambda**: self.main\_window.element.close())  
  
 action = self.main\_window.element.findChild(QAction, **"actionAdd\_raster\_layer"**)  
 action.triggered.connect(self.show\_add\_raster\_layer\_window)  
  
 action = self.main\_window.element.findChild(QAction, **"actionAdd\_vector\_layer"**)  
 action.triggered.connect(self.show\_add\_vector\_layer\_window)  
  
 self.main\_window.element.findChild(QAction, **"actionShow\_list\_of\_layers"**).toggled.connect(  
 **lambda** checked: self.show\_layers() **if** checked **else** self.hide\_layers())  
  
 self.main\_window.element.findChild(QAction, **"actionStart\_model"**).triggered.connect(self.show\_launch\_model\_window)  
  
 **def** show\_layers\_context\_menu(self, point):  
 **if** self.main\_window.elements[**"listOfLayers"**].itemAt(point):  
 layer\_name = self.main\_window.elements[**"listOfLayers"**].itemAt(point).text()  
 context\_menu = QMenu()  
  
 info\_action = QAction(**"Видимость"**, context\_menu)  
 info\_action.setCheckable(**True**)  
 info\_action.setChecked(self.map\_view.has\_layer(layer\_name, **True**).is\_visible)  
 info\_action.toggled.connect(**lambda** checked: self.map\_view.set\_visible(layer\_name, **True**) **if** checked **else** self.map\_view.set\_visible(layer\_name, **False**))

remove\_action = QAction(**"Удалить слой"**, context\_menu)  
 remove\_action.triggered.connect(self.remove\_layer)  
  
 context\_menu.addAction(info\_action)context\_menu.addAction(remove\_action)  
  
 point.setY(point.y() + 10)  
 context\_menu.exec(self.main\_window.elements[**"listOfLayers"**].mapToGlobal(point))  
  
 **def** show\_add\_raster\_layer\_window(self):  
 self.add\_layer\_window.show(0)  
  
 **def** show\_add\_vector\_layer\_window(self):  
 self.add\_layer\_window.show(1)  
  
 **def** show\_launch\_model\_window(self):  
 self.launch\_model\_window.show()  
  
  
 **def** update\_layers\_list(self):  
 self.main\_window.element.findChild(QListWidget, **"listOfLayers"**).clear()  
 **for** layer **in** self.map\_view.layers:  
 self.main\_window.element.findChild(QListWidget, **"listOfLayers"**).addItem(layer.name)  
  
 **def** show\_layers\_list(self):  
 width = self.main\_window.element.findChild(QFrame, **"left\_side\_content"**).width()  
  
 **if** width == 30:  
 newWidth = 300  
 self.main\_window.element.findChild(QLabel, **"label"**).setText(**"Слои"**)  
 **else**:  
 newWidth = 30  
 self.main\_window.element.findChild(QLabel, **"label"**).setText(**""**)  
  
 self.main\_window.element.animation = QPropertyAnimation(self.main\_window  
 .element.findChild(QFrame, **"left\_side\_content"**),  
 **b"minimumWidth"**)  
 self.main\_window.element.animation.setStartValue(width)  
 self.main\_window.element.animation.setEndValue(newWidth)  
  
 self.main\_window.element.animation\_group = QParallelAnimationGroup()  
 self.main\_window.element.animation\_group.addAnimation(self.main\_window.element.animation)  
 self.main\_window.element.animation\_group.start()  
  
 **def** show\_message(self, string, caption, icon, parent=**None**):  
 **if** parent **is None**:  
 parent = self.main\_window.element  
 message\_box = QMessageBox(parent)  
 message\_box.setIcon(icon)  
 message\_box.setText(caption)  
 message\_box.setInformativeText(string)  
 message\_box.setWindowTitle(caption)  
 message\_box.exec\_()  
  
 **def** show\_layers(self):  
 self.main\_window.element.findChild(QAction, **"actionShow\_list\_of\_layers"**).setChecked(**True**)  
 self.show\_layers\_list()  
  
 **def** hide\_layers(self):  
 self.main\_window.element.findChild(QAction, **"actionShow\_list\_of\_layers"**).setChecked(**False**)  
 self.show\_layers\_list()  
  
 **def** remove\_layer(self):  
 **if** self.main\_window.element.findChild(QListWidget, **"listOfLayers"**).currentItem() **is not None**:  
 self.map\_view.remove\_layer(  
 self.main\_window.element.findChild(QListWidget, **"listOfLayers"**).currentItem().text())  
 self.update\_layers\_list()  
  
 **def** new\_project(self):  
 **if** self.main\_window.element.findChild(QFrame, **"left\_side\_content"**).width() > 30:  
 self.hide\_layers()  
  
 self.add\_layer\_window.hide()  
 self.main\_window.elements[**"mainContent"**].removeWidget(self.web)  
 self.web.deleteLater()  
  
 self.web = QWebEngineView()  
 self.main\_window.elements[**"mainContent"**].addWidget(self.web, stretch=1)  
 self.map\_view = MapView(self.web, ui=self)  
 self.update\_layers\_list()

* 1. **Файл UIElements**

**class** UIElement:  
 **def** \_\_init\_\_(self, elements, ui\_path, element\_type):  
 self.element = element\_type  
 uic.loadUi(ui\_path, self.element)  
 self.elements = dict()  
 **for** element **in** elements:  
 self.elements[element[1]] = self.element.findChild(element[0], element[1])

**class** AddLayerWindow(UIElement):  
 OBJECTS = [(QPushButton, **"add\_raster\_btn"**), (QPushButton, **"choose\_raster\_file\_btn"**),  
 (QDoubleSpinBox, **"east\_border"**), (QDoubleSpinBox, **"north\_border"**),  
 (QDoubleSpinBox, **"west\_border"**), (QDoubleSpinBox, **"south\_border"**),  
 (QLineEdit, **"raster\_layer\_name"**), (QLineEdit, **"raster\_path"**),  
 (QPushButton, **"add\_vector\_btn"**), (QPushButton, **"choose\_vector\_file\_btn"**),  
 (QLineEdit, **"vector\_layer\_name"**), (QLineEdit, **"vector\_path"**), (QTabWidget, **"layers\_tab"**)]  
  
 **def** \_\_init\_\_(self, ui\_path, parent, ui):  
 self.parent = parent  
 self.ui = ui  
 super().\_\_init\_\_(AddLayerWindow.OBJECTS, ui\_path, QDialog(self.parent.element))  
  
 **def** initialize(self):  
 self.elements[**"choose\_vector\_file\_btn"**].clicked.connect(self.open\_vector\_file)  
 self.elements[**"add\_vector\_btn"**].clicked.connect(self.add\_vector\_layer)  
 self.elements[**"choose\_raster\_file\_btn"**].clicked.connect(self.open\_raster\_file)  
 self.elements[**"add\_raster\_btn"**].clicked.connect(self.add\_raster\_layer)  
  
 **def** open\_vector\_file(self):  
 options = QFileDialog.Options()  
 file\_name, \_ = QFileDialog.getOpenFileName(self.parent.element, **"Открыть файл"**, **""**,  
 **"GeoJSON, Shapefile (\*.geojson \*.shp)"**, options=options)  
 **if** file\_name:  
 self.elements[**"vector\_path"**].setText(file\_name)  
  
 **def** open\_raster\_file(self):  
 options = QFileDialog.Options()  
 file\_name, \_ = QFileDialog.getOpenFileName(self.parent.element, **"Открыть файл"**, **""**,  
 **"IMG (\*.jpeg \*.jpg \*.tif \*.bmp \*.png)"**, options=options)  
 **if** file\_name:  
 self.elements[**"raster\_path"**].setText(file\_name)  
  
 **def** show(self, tab=0):  
 self.elements[**"layers\_tab"**].setCurrentIndex(tab)  
 self.elements[**"vector\_layer\_name"**].setText(**""**)  
 self.elements[**"vector\_path"**].setText(**""**)  
 self.elements[**"raster\_layer\_name"**].setText(**""**)  
 self.elements[**"raster\_path"**].setText(**""**)  
 self.elements[**"north\_border"**].setValue(0.0)  
 self.elements[**"south\_border"**].setValue(0.0)  
 self.elements[**"east\_border"**].setValue(0.0)  
 self.elements[**"west\_border"**].setValue(0.0)  
 self.element.show()  
  
 **def** hide(self):  
 self.element.hide()  
  
 **def** add\_raster\_layer(self):  
 **try**:  
 self.ui.map\_view.add\_raster\_layer(self.elements[**'raster\_layer\_name'**].text(),  
 self.elements[**'raster\_path'**].text(),  
 (self.elements[**"south\_border"**].value(),  
 self.elements[**"west\_border"**].value()),  
 (self.elements[**"north\_border"**].value(),  
 self.elements[**"east\_border"**].value()))  
 **except** FileOpeningException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка!"**, QMessageBox.Critical, self.element)  
  
 **except** LayerAddingException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка!"**, QMessageBox.Critical, self.element)  
  
 **else**:  
 self.hide()  
 self.ui.update\_layers\_list()  
  
 **def** add\_vector\_layer(self):  
 **try**:  
 self.ui.map\_view.add\_vector\_layer(self.elements[**'vector\_layer\_name'**].text(),  
 self.elements[**'vector\_path'**].text())  
 **except** FileOpeningException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка!"**, QMessageBox.Critical, self.element)  
 **except** LayerAddingException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка!"**, QMessageBox.Critical, self.element)  
 **else**:  
 self.hide()  
 self.ui.update\_layers\_list()

**class** ModelSettingsWindow(UIElement):  
 OBJECTS = [(QPushButton, **"choose\_buildings\_btn"**),  
 (QDoubleSpinBox, **"east\_border"**), (QDoubleSpinBox, **"north\_border"**),  
 (QDoubleSpinBox, **"west\_border"**), (QDoubleSpinBox, **"south\_border"**),  
 (QLineEdit, **"buildings\_path"**), (QDateTimeEdit, **"end"**), (QDateTimeEdit, **"start"**),  
 (QPushButton, **"choose\_fuel\_codes\_btn"**), (QPushButton, **"choose\_fuel\_data\_btn"**),  
 (QPushButton, **"choose\_ignition\_btn"**), (QPushButton, **"choose\_relief\_data\_btn"**),  
 (QPushButton, **"choose\_weather\_btn"**), (QLineEdit, **"fuel\_codes\_path"**),  
 (QLineEdit, **"fuel\_path"**), (QComboBox, **"house\_material"**), (QLineEdit, **"ignition\_path"**),  
 (QLineEdit, **"relief\_path"**), (QSpinBox, **"side"**), (QPushButton, **"start\_model"**),  
 (QSpinBox, **"weatherStep"**), (QLineEdit, **"weather\_path"**), (QLineEdit, **"layer\_name"**)]  
  
 **def** \_\_init\_\_(self, ui\_path, parent, ui):  
 self.parent = parent  
 self.ui = ui  
 self.material = 1.0  
 super().\_\_init\_\_(ModelSettingsWindow.OBJECTS, ui\_path, QDialog(self.parent.element))  
  
 **def** initialize(self):  
 self.elements[**"start\_model"**].clicked.connect(self.launch\_model)  
 self.elements[**"choose\_buildings\_btn"**].clicked.connect(self.open\_osm\_file)  
 self.elements[**"choose\_fuel\_codes\_btn"**].clicked.connect(**lambda** \_: self.open\_text\_file(**"fuel\_codes\_path"**))  
 self.elements[**"choose\_fuel\_data\_btn"**].clicked.connect(**lambda** \_: self.open\_raster\_file(**"fuel\_path"**))  
 self.elements[**"choose\_ignition\_btn"**].clicked.connect(self.open\_vector\_file)  
 self.elements[**"choose\_relief\_data\_btn"**].clicked.connect(**lambda** \_: self.open\_raster\_file(**"relief\_path"**))  
 self.elements[**"choose\_weather\_btn"**].clicked.connect(**lambda** \_: self.open\_text\_file(**"weather\_path"**))  
  
 **def** show(self):  
 self.elements[**"buildings\_path"**].setText(**""**)  
 self.elements[**"fuel\_codes\_path"**].setText(**""**)  
 self.elements[**"fuel\_path"**].setText(**""**)  
 self.elements[**"ignition\_path"**].setText(**""**)  
 self.elements[**"relief\_path"**].setText(**""**)  
 self.elements[**"weather\_path"**].setText(**""**)  
 self.elements[**"layer\_name"**].setText(**""**)  
  
 self.elements[**"north\_border"**].setValue(0.0)  
 self.elements[**"south\_border"**].setValue(0.0)  
 self.elements[**"east\_border"**].setValue(0.0)  
 self.elements[**"west\_border"**].setValue(0.0)  
  
 self.elements[**"side"**].setValue(30)  
 self.elements[**"weatherStep"**].setValue(60)  
 self.element.show()  
  
 **def** open\_text\_file(self, file\_path):  
 options = QFileDialog.Options()  
 file\_name, \_ = QFileDialog.getOpenFileName(self.parent.element, **"Открыть файл"**, **""**,  
 **"CSV (\*.csv)"**, options=options)  
 **if** file\_name:  
 self.elements[file\_path].setText(file\_name)  
  
 **def** open\_raster\_file(self, file\_path):  
 options = QFileDialog.Options()  
 file\_name, \_ = QFileDialog.getOpenFileName(self.parent.element, **"Открыть файл"**, **""**,  
 **"GeoTiff (\*.tif \*.asc)"**, options=options)  
 **if** file\_name:  
 self.elements[file\_path].setText(file\_name)  
  
 **def** open\_vector\_file(self):  
 options = QFileDialog.Options()  
 file\_name, \_ = QFileDialog.getOpenFileName(self.parent.element, **"Открыть файл"**, **""**,  
 **"Shapefile (\*.shp)"**, options=options)  
 **if** file\_name:  
 self.elements[**"ignition\_path"**].setText(file\_name)  
  
 **def** open\_osm\_file(self):  
 options = QFileDialog.Options()  
 file\_name, \_ = QFileDialog.getOpenFileName(self.parent.element, **"Открыть файл"**, **""**,  
 **"OSM, Shapefile (\*.osm \*.shp)"**, options=options)  
 **if** file\_name:  
 self.elements[**"buildings\_path"**].setText(file\_name)  
  
 @staticmethod  
 **def** jarWrapper(\*args):  
process = check\_output([**'java'**, **'-Djava.library.path=..\model\lib-gdal'**, **'-jar'**] + list(args), stderr=PIPE)  
  
 **def** launch\_model(self):  
 **try**:  
 **if** self.elements[**"house\_material"**].currentText() == **"Смешанный"**:  
 self.material = 0.8  
 **elif** self.elements[**"house\_material"**].currentText() == **"Огнеупорный"**:  
 self.material = 0.6  
 **else**:  
 self.material = 1.0**if not** self.check\_model\_params():  
 args = [**'../model/model-1.0.jar'**, str(self.elements[**"side"**].value()),  
 self.elements[**"relief\_path"**].text(), self.elements[**"fuel\_path"**].text(),  
 self.elements[**"fuel\_codes\_path"**].text(), self.elements[**"weather\_path"**].text(),  
 self.elements[**"ignition\_path"**].text(),  
 str(self.elements[**"start"**].dateTime().toString(**"yyyy.MM.dd HH:mm"**)).split()[0],  
 str(self.elements[**"start"**].dateTime().toString(**"yyyy.MM.dd HH:mm"**)).split()[1],  
 str(self.elements[**"end"**].dateTime().toString(**"yyyy.MM.dd HH:mm"**)).split()[0],  
 str(self.elements[**"end"**].dateTime().toString(**"yyyy.MM.dd HH:mm"**)).split()[1],  
 str(self.elements[**"weatherStep"**].value()),  
 str(self.material), str(self.elements[**"north\_border"**].value()),  
 str(self.elements[**"west\_border"**].value()),  
 str(self.elements[**"south\_border"**].value()), str(self.elements[**"east\_border"**].value()),  
 self.elements[**"buildings\_path"**].text()]  
  
 self.jarWrapper(\*args)  
  
 self.ui.show\_message(**"Моделирование прошло успешно"**, **"Успешно"**, QMessageBox.Information)  
 **except** CalledProcessError:  
 self.ui.show\_message(**"Ошибка при моделировании"**, **"Ошибка"**, QMessageBox.Critical)  
 **except** LayerAddingException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка"**, QMessageBox.Critical)  
 **except** FileOpeningException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка"**, QMessageBox.Critical)  
 **except** ModelSettingException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка"**, QMessageBox.Critical)  
 **else**:  
 self.hide()  
  
 **try**:  
 self.ui.map\_view.add\_raster\_layer(self.elements[**"layer\_name"**].text(),  
 **"../data/result/result\_"** + str(self.elements[**"end"**]  
 .dateTime().toString(  
 **"yyyy\_MM\_dd\_HH\_mm"**)) + **".tif"**,  
 (self.elements[**"south\_border"**].value(),  
 self.elements[**"west\_border"**].value()),  
 (self.elements[**"north\_border"**].value(),  
 self.elements[**"east\_border"**].value()),  
 color=gdal.GCI\_RedBand)  
 self.ui.update\_layers\_list()  
 **except** FileOpeningException **as** ex:  
 self.ui.show\_message(ex.message, **"Ошибка"**, QMessageBox.Critical, self.element)  
  
 **def** hide(self):  
 self.element.hide()  
  
 **def** check\_file(self, file\_name):  
 ds = gdal.Open(self.elements[**"relief\_path"**].text(), gdal.GA\_ReadOnly)  
 **if not** ds:  
 **raise** FileOpeningException(**"Ошибка при открытии файла"** + file\_name + **"!"**)  
 **else**:  
 ds = **None  
  
 def** check\_model\_params(self):  
 self.check\_file(self.elements[**"relief\_path"**].text())  
 self.check\_file(self.elements[**"fuel\_path"**].text())  
  
 **if not** path.exists(self.elements[**"fuel\_codes\_path"**].text()):  
 **raise** FileOpeningException(**"Файла кодов топлива по данному пути не существует"**)  
  
 **if not** path.exists(self.elements[**"weather\_path"**].text()):  
 **raise** FileOpeningException(**"Файла погоды по данному пути не существует"**)  
  
 **if not** path.exists(self.elements[**"ignition\_path"**].text()):  
 **raise** FileOpeningException(**"Файла территории начального возгорания"  
 " по данному пути не существует"**)  
  
 *# Даты не должны заходить одна за другую.* **if** self.elements[**"start"**].dateTime().secsTo(self.elements[**"end"**].dateTime()) <= 0:  
 **raise** ModelSettingException(**"Дата окончания должна быть позже даты начала"**)  
  
 *# Расположение координат.* **if** self.elements[**"south\_border"**].value() > self.elements[**"north\_border"**].value():  
 **raise** ModelSettingException(**"Южная граница не может быть выше северной!"**)  
 **if** self.elements[**"west\_border"**].value() > self.elements[**"east\_border"**].value():  
 **raise** ModelSettingException(**"Западная граница не может быть левее восточной!"**)  
  
 **if not** self.ui.map\_view.check\_layer\_name(self.elements[**"layer\_name"**].text()):  
 **raise** LayerAddingException(**"Некорректное название слоя!"**)  
  
 **if** self.ui.map\_view.has\_layer(self.elements[**"layer\_name"**].text()):  
 **raise** LayerAddingException(**"Слой с таким названием уже существует"**)

**return** 0

* 1. **Файл MapView.py**

**import** os  
**import** geojson  
**from** osgeo **import** gdal  
  
**from** Components.Exceptions **import** LayerAddingException, FileOpeningException, MapCreatingException, \  
 LayerNotFoundException  
**from** Components.Layers **import** RasterLayer, VectorLayer  
**from** Components.MapEngine **import** DEFAULT\_HTML, OSM\_TILE\_CREATION\_SCRIPT, \  
 MAP\_CREATION\_SCRIPT, ADD\_TILE\_TO\_MAP\_SCRIPT, RASTER\_LAYER\_CREATION\_SCRIPT, GEOJSON\_LAYER\_CREATION\_SCRIPT, \  
 GEOJSON\_LAYER\_ADD\_DATA\_SCRIPT, SHOW\_LAYER\_SCRIPT, HIDE\_LAYER\_SCRIPT, \  
 REMOVE\_LAYER\_SCRIPT  
**from** Components.Utilities **import** image\_to\_data, shp\_to\_json  
  
  
**class** MapView:  
 TILES\_STRING\_TO\_SCRIPT = {**"OpenStreetMap"**: OSM\_TILE\_CREATION\_SCRIPT}  
  
 **def** \_\_init\_\_(self, window, map\_tiles=**"OpenStreetMap"**, save\_file\_path=**None**, ui=**None**):  
 **if** map\_tiles **not in** MapView.TILES\_STRING\_TO\_SCRIPT.keys(): *# ["OpenStreetMap", "Mapbox Bright", "Mapbox Control Room", "Stamen"]:* **raise** Exception(**"Undefined map tiles"**)  
 self.layers = []  
 self.save\_file\_path = save\_file\_path  
 self.ui = ui  
 self.map\_tiles = map\_tiles  
 self.window = window  
 self.window.setHtml(DEFAULT\_HTML)  
 self.window.loadFinished.connect(self.on\_load\_finished)  
  
 **def** on\_load\_finished(self, status):  
 **if** status:  
 **if** self.save\_file\_path **is None**:  
 self.window.page().runJavaScript(MAP\_CREATION\_SCRIPT + MapView.TILES\_STRING\_TO\_SCRIPT[self.map\_tiles] +  
 ADD\_TILE\_TO\_MAP\_SCRIPT)  
  
 **def** has\_layer(self, layer\_name, return\_layer=**False**):  
 index = -1  
 **for** i **in** range(0, len(self.layers)):  
 **if** self.layers[i].name == layer\_name:  
 index = i  
 **if** return\_layer:  
 **if** index == -1:  
 **return None  
 else**:  
 **return** self.layers[index]  
 **else**:  
 **return** index != -1  
  
 **def** add\_map\_layer(self, layer\_name, map\_type):  
 **if** self.has\_layer(layer\_name):  
 **raise** LayerAddingException(**"Слой с таким названием уже существует!"**)  
  
 **def** add\_raster\_layer(self, layer\_name, file\_path, upper\_left\_bound, lower\_right\_bound,  
 color=gdal.GCI\_GrayIndex, data=**None**):  
 **if not** self.check\_layer\_name(layer\_name):  
 **raise** LayerAddingException(**"Некорректное название слоя!"**)  
 **if** self.has\_layer(layer\_name):  
 **raise** LayerAddingException(**"Слой с таким названием уже существует"**)  
  
 **if** lower\_right\_bound[0] < upper\_left\_bound[0]:  
 **raise** LayerAddingException(**"Южная граница не может быть выше северной!"**)  
 **if** lower\_right\_bound[1] < upper\_left\_bound[1]:  
 **raise** LayerAddingException(**"Западная граница не может быть левее восточной!"**)  
  
 **if** data **is None and not** os.path.exists(file\_path):  
 **raise** FileOpeningException(**"Файл не найден!"**)  
 **else**:  
 bounds = [upper\_left\_bound, lower\_right\_bound]  
 string\_bounds = **"[["** + str(bounds[0][0]) + **", "** + str(bounds[0][1]) + **"], ["** + \  
 str(bounds[1][0]) + **", "** + str(bounds[1][1]) + **"]]"  
 if** data **is None**:  
 data = image\_to\_data(file\_path, color)  
 self.window.page().runJavaScript(**"var createLayerData = '"** + data + **"';\n"** +  
 RASTER\_LAYER\_CREATION\_SCRIPT % (layer\_name, string\_bounds, layer\_name))  
 self.layers.append(RasterLayer(layer\_name, data, bounds))  
  
 **def** add\_vector\_layer(self, layer\_name, path, data=**None**):  
 **global** geo\_data  
 **if not** self.check\_layer\_name(layer\_name):  
 **raise** LayerAddingException(**"Некорректное название слоя"**)  
 **if** self.has\_layer(layer\_name):  
 **raise** LayerAddingException(**"Слой с таким названием уже существует!"**)  
 **if not** os.path.exists(path):  
 **raise** FileOpeningException(**"Файл не найден!"**)  
  
 **if** data **is None**:  
 **try**:  
 file\_format = os.path.splitext(path)[-1][1:]  
  
 **if** file\_format == **"shp"**:  
 path = shp\_to\_json(path)  
  
 geo\_file = open(path, **'r'**)  
 geo\_data = geojson.load(geo\_file)  
 geo\_file.close()  
 **except** Exception:  
 **raise** FileOpeningException(**"Невозможно прочитать файл!"**)  
 **else**:  
 self.layers.append(VectorLayer(layer\_name, str(geo\_data)))  
 self.window.page().runJavaScript(GEOJSON\_LAYER\_CREATION\_SCRIPT % (layer\_name, layer\_name))  
 self.window.page().runJavaScript(GEOJSON\_LAYER\_ADD\_DATA\_SCRIPT % (layer\_name, str(geo\_data)))  
 **else**:  
 self.layers.append(VectorLayer(layer\_name, data))  
 self.window.page().runJavaScript(GEOJSON\_LAYER\_CREATION\_SCRIPT % (layer\_name, layer\_name))  
 self.window.page().runJavaScript(GEOJSON\_LAYER\_ADD\_DATA\_SCRIPT % (layer\_name, str(data)))  
  
 @staticmethod  
 **def** check\_layer\_name(layer\_name):  
 layer\_name = layer\_name.replace(**" "**, **""**)  
 **return** len(layer\_name) > 0  
  
 **def** remove\_layer(self, layer\_name):  
 layer = self.has\_layer(layer\_name, **True**)  
 **if** layer **is None**:  
 **raise** LayerNotFoundException(**"Слой не найден"**)  
 self.layers.remove(layer)  
 self.window.page().runJavaScript(REMOVE\_LAYER\_SCRIPT % (layer\_name, layer\_name))  
  
 **def** set\_visible(self, layer\_name, is\_visible):  
 layer = self.has\_layer(layer\_name, **True**)  
 **if** layer **is None**:  
 **raise** LayerNotFoundException(**"Слой не найден"**)  
 layer.is\_visible = is\_visible  
 **if** layer.is\_visible:  
 self.window.page().runJavaScript(SHOW\_LAYER\_SCRIPT % (layer\_name, layer\_name))  
 **else**:  
 self.window.page().runJavaScript(HIDE\_LAYER\_SCRIPT % (layer\_name, layer\_name))

* 1. **Файл MapEngine.py**

DEFAULT\_HTML = **"""  
 <!DOCTYPE html>  
<head>   
 <meta http-equiv="content-type" content="text/html; charset=UTF-8" />  
  
 <script>  
 L\_NO\_TOUCH = false;  
 L\_DISABLE\_3D = false;  
 </script>  
  
 <script src="https://cdn.jsdelivr.net/npm/leaflet@1.5.1/dist/leaflet.js"></script>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>  
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/Leaflet.awesome-markers/2.0.2/leaflet.awesome-markers.js"></script>  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/leaflet@1.5.1/dist/leaflet.css"/>  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"/>  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css"/>  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.3/css/font-awesome.min.css"/>  
 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/Leaflet.awesome-markers/2.0.2/leaflet.awesome-markers.css"/>  
 <link rel="stylesheet" href="https://rawcdn.githack.com/python-visualization/folium/master/folium/templates/leaflet.awesome.rotate.css"/>  
 <style>html, body {width: 100%;height: 100%;margin: 0;padding: 0;}</style>  
 <style>#map {position:absolute;top:0;bottom:0;right:0;left:0;}</style>  
  
 <meta name="viewport" content="width=device-width,  
 initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />  
 <style>  
 #main-map {  
 position: relative;  
 width: 100.0%;  
 height: 100.0%;  
 left: 0.0%;  
 top: 0.0%;  
 }  
 </style>  
  
</head>  
<body>   
  
 <div class="folium-map" id="main-map"></div>  
  
</body>  
<script>  
 var layers = {};  
</script>  
"""**MAP\_CREATION\_SCRIPT = **"""  
 var mainMap = L.map("main-map",  
 {  
 center: [34.11, -118.5],  
 crs: L.CRS.EPSG3857,  
 zoom: 10,  
 zoomControl: true,  
 preferCanvas: false,  
 }  
 );  
"""**OSM\_TILE\_CREATION\_SCRIPT = **"""  
 var mapTileLayer = L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",  
 {  
 "attribution": "Data by &copy; <a href='http://openstreetmap.org'>OpenStreetMap</a>, under <a href='http://www.openstreetmap.org/copyright'>ODbL</a>.",  
 "detectRetina": false,  
 "maxNativeZoom": 18,  
 "maxZoom": 18,  
 "minZoom": 0,  
 "noWrap": false,  
 "opacity": 1,  
 "subdomains": "abc",  
 "tms": false  
 }  
 );  
"""**ADD\_TILE\_TO\_MAP\_SCRIPT = **"""  
 mapTileLayer.addTo(mainMap);  
"""**GEOJSON\_LAYER\_CREATION\_SCRIPT = **"""  
 layers["%s"] = L.geoJson(null, { onEachFeature: (feature, layer) => { layer.on({ click: function(e) { mainMap.fitBounds(e.target.getBounds()); }}); }});  
 layers["%s"].addTo(mainMap);  
"""**GEOJSON\_LAYER\_ADD\_DATA\_SCRIPT = **"""  
 layers["%s"].addData(%s);  
"""**RASTER\_LAYER\_CREATION\_SCRIPT = **"""  
 layers["%s"] = L.imageOverlay(createLayerData, %s);  
 layers["%s"].addTo(mainMap);  
"""**REMOVE\_LAYER\_SCRIPT = **"""  
 mainMap.removeLayer(layers["%s"]);  
 delete layers["%s"];  
"""**SHOW\_LAYER\_SCRIPT = **"""  
 if (!mainMap.hasLayer(layers["%s"])) {  
 layers["%s"].addTo(mainMap);  
 }  
"""**HIDE\_LAYER\_SCRIPT = **"""  
 if (mainMap.hasLayer(layers["%s"])) {  
 mainMap.removeLayer(layers["%s"]);  
 }  
"""**

* 1. **Файл Layers.py**

**from** Components.Exceptions **import** LayerCreatingException  
  
  
**class** Layer:  
 **def** \_\_init\_\_(self, name, layer\_type):  
 **if** layer\_type **not in** [**'raster'**, **'vector'**]:  
 **raise** LayerCreatingException(**"undefined type of layer"**)  
 self.name = name  
 self.type = layer\_type  
 self.is\_visible = **True  
  
  
class** RasterLayer(Layer):  
 **def** \_\_init\_\_(self, name, data, bounds):  
 super().\_\_init\_\_(name, **"raster"**)  
 self.data = data  
 self.bounds = bounds  
  
  
**class** VectorLayer(Layer):  
 **def** \_\_init\_\_(self, name, data):  
 super().\_\_init\_\_(name, **"vector"**)  
 self.data = data

* 1. **Файл Utilities.py**

**from** Components.Exceptions **import** LayerAddingException  
  
\_VALID\_URLS = set(uses\_relative + uses\_netloc + uses\_params)  
\_VALID\_URLS.discard(**''**)  
  
  
**def** \_is\_url(url):  
 **try**:  
 **return** urlparse(url).scheme **in** \_VALID\_URLS  
 **except** Exception:  
 **return False  
  
  
def** write\_png(data, color = gdal.GCI\_GrayIndex):  
 ds = gdal.Open(data)  
  
 **if** ds.RasterCount > 1:  
 **raise** LayerAddingException(**"Растровый .tif файл не должен "  
 "содержать более 1 слоя!"**)  
 band = ds.GetRasterBand(1)  
  
 root\_ext = os.path.splitext(data)[0]  
 output\_path = os.path.basename(root\_ext) + **".png"** band.SetRasterColorInterpretation(color)  
 band.SetNoDataValue(0)  
 stats = band.GetStatistics(**True**, **True**)  
  
 gdal.Translate(output\_path, ds, format=**"PNG"**,  
 scaleParams=[[min(stats), max(stats), [0, 255]]])  
 **del** band  
 **del** ds  
 **return** output\_path  
  
  
**def** image\_to\_data(path, color = gdal.GCI\_GrayIndex):  
 **if** isinstance(path, str) **and not** \_is\_url(path):  
 file\_format = os.path.splitext(path)[-1][1:]  
  
 **if** file\_format == **"tif"**:  
 png\_path = write\_png(path, color)  
 **with** io.open(png\_path, **'rb'**) **as** f:  
 img = f.read()  
 b64encoded = base64.b64encode(img).decode(**'utf-8'**)  
 url = **'data:image/png;base64,{}'**.format(b64encoded)  
 **else**:  
 **with** io.open(path, **'rb'**) **as** f:  
 img = f.read()  
 b64encoded = base64.b64encode(img).decode(**'utf-8'**)  
 url = **'data:image/{};base64,{}'**.format(file\_format, b64encoded)  
  
 **return** url.replace(**'\n'**, **' '**)  
 **else**:  
 url = json.loads(json.dumps(path))  
 **return** url.replace(**'\n'**, **' '**)  
  
**def** shp\_to\_json(path):  
 root\_ext = os.path.splitext(path)[0]  
 output\_path = os.path.basename(root\_ext) + **".geojson"** reader = shapefile.Reader(path)  
 fields = reader.fields[1:]  
 field\_names = [field[0] **for** field **in** fields]  
 buffer = []  
 **for** sr **in** reader.shapeRecords():  
 atr = dict(zip(field\_names, sr.record))  
 geom = sr.shape.\_\_geo\_interface\_\_  
 buffer.append(dict(type=**"Feature"**,  
 geometry=geom, properties=atr))  
  
 geojson = open(output\_path, **"w"**)  
 **try**:  
 geojson.write(dumps({**"type"**: **"FeatureCollection"**, **"features"**: buffer}, indent=2, default=str) + **"\n"**)  
 **except** TypeError:  
 print(**"Hi!"**)  
 geojson.close()  
 **return** output\_path

* 1. **Файл Layers.py**

**from** Components.Exceptions **import** LayerCreatingException  
  
**class** Layer:  
 **def** \_\_init\_\_(self, name, layer\_type):  
 **if** layer\_type **not in** [**'raster'**, **'vector'**]:  
 **raise** LayerCreatingException(**"undefined type of layer"**)  
 self.name = name  
 self.type = layer\_type  
 self.is\_visible = **True  
  
  
class** RasterLayer(Layer):  
 **def** \_\_init\_\_(self, name, data, bounds):  
 super().\_\_init\_\_(name, **"raster"**)  
 self.data = data  
 self.bounds = bounds  
  
  
**class** VectorLayer(Layer):  
 **def** \_\_init\_\_(self, name, data):  
 super().\_\_init\_\_(name, **"vector"**)  
 self.data = data

* 1. **Файл Exceptions.py**

**class** LayerAddingException(Exception):  
 **def** \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
 self.message = message  
  
  
**class** LayerCreatingException(Exception):  
 **def** \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
 self.message = message  
  
  
**class** MapCreatingException(Exception):  
 **def** \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
 self.message = message  
  
  
**class** FileOpeningException(Exception):  
 **def** \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
 self.message = message  
  
  
**class** NotVectorLayerException(Exception):  
 **def** \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
 self.message = message  
  
  
**class** LayerNotFoundException(Exception):  
 **def** \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
 self.message = message  
  
  
**class** ModelSettingException(Exception):  
 **def** \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
 self.message = message

# **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |