Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Московский государственный технический университет им. Н. Э. Баумана

23 0102

АСОИ поиска алгоритмов распознавания изоморфизма графов с помощью генетического программирования Исходный код

Студент группы ИУ5-82
___________ Гуща А. В
“____” ____________ ________

2016

Содержание

1 Пакет APP(MAIN)

```
module app;

import gtk.Builder;
import gtk.Button;
import gtk.Main;
import gtk.Widget;
import gtk.ApplicationWindow;
import gtk.MenuItem;
import gobject.Type;

import std.stdio;
import std.getopt;
import std.c.process;
import std.file;

import dlogg.strict;

import gui.evolution;
import gui.results;
import gui.settings;

import project;
import application;

enum helpMsg =
"graph-isomorph [options]

options:  --gui=<path>  - path to glade file. Optional, default is 'gui.glade'.
          --log=<path>  - path to log file. Optional, default is 'graph-isomorph.log'.
          --proj=<path> - path to project file. Optional, default is '"~Project.defaultProjectPath~"'.
          --help        - display the message.";

void main(string[] args)
{
        string gladeFile = "./gui.glade";
        string logFile = "./graph-isomorph.log";
        string projFile = Project.defaultProjectPath;

        bool help = false;
        getopt(args,
            "gui", &gladeFile,
            "log", &logFile,
            "proj", &projFile,
            "help", &help
        );

        if(help)
        {
            writeln(helpMsg);
            return;
        }

        Main.initMultiThread(args);

        auto application = new Application(logFile, gladeFile, projFile);
        scope(exit) application.finalize();

        Main.run();
}
```

2 Пакет APPLICATION

```
module application;

import gtk.Builder;
import gtk.ApplicationWindow;

import gui.settings;
import gui.evolution;
import gui.results;

import dlogg.strict;

import project;
import std.file;
```

```
class Application
{
    shared ILogger logger;

    SettingsWindow settingsWindow;
    EvolutionWindow evolutionWindow;
    ResultsWindow resultsWindow;

    Project project;

    this(string logFile, string gladeFile, string projFile)
    {
        logger = new shared StrictLogger(logFile);

        logger.logInfo("Loading project file...");
        project = new Project(logger);
        if(projFile.exists)
        {
            project.open(projFile);
        }
        else
        {
            logger.logInfo("Cannot find project file, creating new project");
        }

        Builder builder = new Builder();
        if( !builder.addFromFile(gladeFile) )
        {
            logger.logError(text("Failed to create gui from glade file '", gladeFile, "'!"));
            return;
        }

        logger.logInfo("Loading settings window");
        auto settingsWnd = cast(ApplicationWindow)builder.getObject("SettingsWindow");
        if(settingsWnd is null)
        {
            logger.logError("Failed to create settings window!");
            return;
        }

        logger.logInfo("Loading evolution window");
        auto evolutionWnd = cast(ApplicationWindow)builder.getObject("EvolutionWindow");
        if(evolutionWnd is null)
        {
            logger.logError("Failed to create evolution window!");
            return;
        }

        logger.logInfo("Loading results window");
        auto resultsWnd = cast(ApplicationWindow)builder.getObject("ResultsWindow");
        if(resultsWnd is null)
        {
            logger.logError("Failed to create results window!");
            return;
        }

        settingsWindow = new SettingsWindow(this, builder, logger, project, settingsWnd, evolutionWnd, resultsWnd);
        evolutionWindow = new EvolutionWindow(this, builder, logger, project, settingsWnd, evolutionWnd, resultsWnd
            ↪ );
        resultsWindow = new ResultsWindow(this, builder, logger, project, settingsWnd, evolutionWnd, resultsWnd);

        updateAll();
    }

    void updateAll()
    {
        settingsWindow.updateContent();
        evolutionWindow.updateContent();
        resultsWindow.updateContent();
    }

    void finalize()
    {
        logger.finalize();
    }
}
```

3 Пакет PROJECT

```
module project;
```

```
import evol.progtype;
import evol.compiler;

import dyaml.all;
import std.path;
import std.file;
import std.stream;

import dlogg.log;

class Project
{
    ProgramType programType;
    GraphPopulation population;
    string name;
    string filename;
    string populationPath;

    bool popLoaded = true;

    private shared ILogger logger;

    enum defaultProjectPath = "./project.yaml";
    enum evolSettings = "evolutionSettings";
    enum projectName = "name";
    enum popPathKeyName = "population";

    this(shared ILogger logger)
    {
        this.logger = logger;
        programType = new ProgramType();
        filename = defaultProjectPath;
        name = defaultProjectPath.baseName;
    }

    private void open(Node root)
    {
        programType = new ProgramType();

        if(root.containsKey(projectName))
        {
            name = root[projectName].as!string;
        }

        if(root.containsKey(evolSettings))
        {
            Node node = root[evolSettings];

            void setValue(string field)()
            {
                mixin("alias T = typeof(programType."~field~");");
                if(node.containsKey(field))
                {
                    mixin("programType."~field~` = node["`~field~`"].as!`~T.stringof~";");
                }
            }

            setValue!"progMinSize";
            setValue!"progMaxSize";
            setValue!"newOpGenChance";
            setValue!"newScopeGenChance";
            setValue!"newLeafGenChance";
            setValue!"scopeMinSize";
            setValue!"scopeMaxSize";
            setValue!"mutationChance";
            setValue!"crossingoverChance";
            setValue!"mutationChangeChance";
            setValue!"mutationReplaceChance";
            setValue!"mutationDeleteChance";
            setValue!"mutationAddLineChance";
            setValue!"mutationRemoveLineChance";
            setValue!"copyingPart";
            setValue!"deleteMutationRiseGenomeSize";
            setValue!"maxGenomeSize";
            setValue!"populationSize";
            setValue!"graphPermuteChance";
            setValue!"graphNodesCountMin";
            setValue!"graphNodesCountMax";
            setValue!"graphLinksCountMin";
            setValue!"graphLinksCountMax";
            setValue!"graphPermutesCountMin";
            setValue!"graphPermutesCountMax";
        }

        if(root.containsKey(popPathKeyName))
        {
            populationPath = root[popPathKeyName].as!string;
        }
    }

    void open(string filename)
    {
        this.filename = filename;
        open(Loader(filename).load);

        loadPopulation();
    }

    void save(string filename)
    {
        this.filename = filename;
        savePopulation();
        Dumper(filename).dump(dump);
    }

    private void savePopulation()
    {
        if(population is null) return;

        if(populationPath == "")
        {
            populationPath = population.name~".yaml";
        }

        try
        {
            if(!populationPath.dirName.exists)
            {
                mkdirRecurse(populationPath.dirName);
            }

            Dumper(populationPath).dump(population.saveYaml);
        } catch(Exception e)
        {
            logger.logError(text("Failed to load population from '", populationPath, "'. Reason: ", e.msg));
            debug logger.logError(e.toString);

            population = null;
            popLoaded = true;
        }
    }

    private void loadPopulation()
    {
        try
        {
            auto node = Loader(populationPath).load();
            population = GraphPopulation.loadYaml(node);
            popLoaded = true;
        } catch(Exception e)
        {
            logger.logError(text("Failed to load population from '", populationPath, "'. Reason: ", e.msg));
            debug logger.logError(e.toString);

            population = null;
            popLoaded = true;
        }
    }

    private Node dump()
    {
        Node[string] emap;

        void setValue(string field)()
        {
            mixin("alias T = typeof(programType."~field~");");
            emap[field] = Node(mixin("programType."~field));
        }

        setValue!"progMinSize";
        setValue!"progMaxSize";
        setValue!"newOpGenChance";
        setValue!"newScopeGenChance";
        setValue!"newLeafGenChance";
        setValue!"scopeMinSize";
        setValue!"scopeMaxSize";
        setValue!"mutationChance";
        setValue!"crossingoverChance";
        setValue!"mutationChangeChance";
        setValue!"mutationReplaceChance";
        setValue!"mutationDeleteChance";
        setValue!"mutationAddLineChance";
        setValue!"mutationRemoveLineChance";
        setValue!"copyingPart";
        setValue!"deleteMutationRiseGenomeSize";
        setValue!"maxGenomeSize";
        setValue!"populationSize";
        setValue!"graphPermuteChance";
        setValue!"graphNodesCountMin";
        setValue!"graphNodesCountMax";
        setValue!"graphLinksCountMin";
        setValue!"graphLinksCountMax";
        setValue!"graphPermutesCountMin";
        setValue!"graphPermutesCountMax";

        return Node([
            projectName     : Node(name),
            evolSettings    : Node(emap),
            popPathKeyName : Node(populationPath)
            ]);
    }

    void recreate(string filename)
    {
        programType = new ProgramType();
        name = filename.baseName;
        this.population = null;
        this.filename = filename;
        save(filename);
    }
}
```

4 Пакет GUI.EVOLUTION

```
module gui.evolution;

import gtk.Builder;
import gtk.MenuItem;
import gtk.ApplicationWindow;
import gtk.Image;
import gtk.ToolButton;
import gtk.ProgressBar;
import gdk.Threads;
import gtk.Entry;

import gui.util;
import gui.generic;
import dlogg.log;

import graph.directed;

import project;
import application;

import std.file;
import std.stdio;
import std.process;
import std.path;
import std.conv;
import std.concurrency;
import std.datetime;
import std.functional;

import core.thread;

import evol.compiler;
import evol.world;

class EvolutionWindow : GenericWindow
{
    this(Application app, Builder builder, shared ILogger logger
        , Project project
        , ApplicationWindow settingsWindow
        , ApplicationWindow evoluitionWindow
        , ApplicationWindow resultsWindow)
    {
        super(app, builder, logger, project, evoluitionWindow);

        evoluitionWindow.hide();
        evoluitionWindow.addOnHide( (w) => onWindowHideShow(AppWindow.Evolution, true) );
        evoluitionWindow.addOnShow( (w) => onWindowHideShow(AppWindow.Evolution, false) );
        evoluitionWindow.addOnDelete( (e, w) { evoluitionWindow.hide; return true; } );

        auto showSettingsWndItem = cast(MenuItem)builder.getObject("ShowSettingsWndItem2");
    if(showSettingsWndItem is null)
    {
        logger.logError("EvolutionWnd: failed to get show settings wnd item!");
        assert(false);
    }
    showSettingsWndItem.addOnActivate( (w) => settingsWindow.showAll() );

    auto showResultsWndItem = cast(MenuItem)builder.getObject("ShowResultsWndItem2");
    if(showResultsWndItem is null)
    {
        logger.logError("EvolutionWnd: failed to get show results wnd item!");
        assert(false);
    }
    showResultsWndItem.addOnActivate( (w) => resultsWindow.showAll() );

    initProjectSaveLoad("2");
    initAboutDialog("2");

    initEvolution();
    initEvolutionControl();
}

void setInputImages(IDirectedGraph first, IDirectedGraph second)
{
    void setImage(IDirectedGraph graph, string wname)
    {
        auto image = cast(Image)builder.getObject(wname);
        assert(image !is null);

        try
        {
            enum tempImageDir = "./images";
            if(!tempImageDir.exists)
            {
                mkdirRecurse(tempImageDir);
            }

            string dotFilename = buildPath(tempImageDir, wname~".dot");
            string imageFilename = buildPath(tempImageDir, wname~".png");

            auto file = File(dotFilename, "w");
            file.writeln(graph.genDot());
            file.close();

            shell(text("dot -Tpng ", dotFilename, " > ", imageFilename));

            image.setFromFile(imageFilename);
        }
        catch(Exception e)
        {
            logger.logError("Failed to load image from graph for "~wname);
            logger.logError(e.msg);
        }
    }

    setImage(first, "InputGraphImage1");
    setImage(second, "InputGraphImage2");
}

void setGenerationNumber(size_t i)
{
    auto entry = cast(Entry)builder.getObject("GenerationNumberEntry");
    assert(entry !is null);

    entry.setText(to!string(i+1));
}

void setMaxFitness(double fitness)
{
    auto entry = cast(Entry)builder.getObject("MaxFitnessEntry");
    assert(entry !is null);

    entry.setText(to!string(fitness));
}

void setAvarageFitness(double fitness)
{
    auto entry = cast(Entry)builder.getObject("AvarageFitnessEntry");
    assert(entry !is null);

    entry.setText(to!string(fitness));
}

void initEvolutionControl()
{
    auto startBtn = cast(ToolButton)builder.getObject("EvolutionStartButton");
```

```
        assert(startBtn !is null);

        startBtn.addOnClicked((b)
        {
            try
            {
                startEvolution();
            } catch(Throwable th)
            {
                logger.logError(th.toString);
            }
        });

        auto pauseBtn = cast(ToolButton)builder.getObject("EvolutionPauseButton");
        assert(pauseBtn !is null);

        pauseBtn.addOnClicked((b)
        {
            try
            {
                pauseEvolution();
            } catch(Throwable th)
            {
                logger.logError(th.toString);
            }
        });

        auto stopBtn = cast(ToolButton)builder.getObject("EvolutionStopButton");
        assert(stopBtn !is null);

        stopBtn.addOnClicked((b)
        {
            try
            {
                stopEvolution();

                auto progressBar = cast(ProgressBar)builder.getObject("EvolutionProgressBar");
                assert(progressBar !is null);
                progressBar.setFraction(0);

            } catch(Throwable th)
            {
                logger.logError(th.toString);
            }
        });
    }

    void initEvolution()
    {
        compiler = new GraphCompiler(
                new GraphCompilation(project, ()
                        {
                                threadsEnter();
                                application.updateAll();
                                threadsLeave();
                        })
                , project.programType
                , new GraphWorld(
                project.programType
                ,(gr1, gr2)
                {
                    threadsEnter();
                    setInputImages(gr1, gr2);
                    threadsLeave();
                }));

        evolState = EvolutionState.Stoped;
    }

    void startEvolution()
    {
        final switch(evolState)
        {
            case(EvolutionState.Running):
            {
                return;
            }
            case(EvolutionState.Paused):
            {
                evolutionTid.send(thisTid, EvolutionCommand.Resume);
                return;
            }
            case(EvolutionState.Stoped):
            {
                compiler.clean();
                if(project.popLoaded)
                {
                    project.population = compiler.addPop(
                        project.programType.populationSize);
                } else
                {
                    project.popLoaded = false;
                }
                evolutionTid = spawn(&evolutionThread, cast(shared)this);
                evolutionTid.send(thisTid);
                return;
            }
        }
    }

    void stopEvolution()
    {
        final switch(evolState)
        {
            case(EvolutionState.Running):
            {
                evolutionTid.send(thisTid, EvolutionCommand.Stop);
                return;
            }
            case(EvolutionState.Paused):
            {
                evolutionTid.send(thisTid, EvolutionCommand.Stop);
                return;
            }
            case(EvolutionState.Stoped):
            {
                return;
            }
        }
    }

    override void updateContent()
    {
        super.updateContent();

        if(project.population !is null)
        {
            setGenerationNumber(cast(size_t)project.population.generation);

            double val = 0.0;
            double maxFitness = 0.0;
            foreach(ind; project.population)
            {
                if(ind.fitness > maxFitness)
                    maxFitness = ind.fitness;

                val += ind.fitness;
            }

            setMaxFitness(maxFitness);
            if(val != 0.0)
            {
                setAvarageFitness(val / cast(double) project.population.length);
            } else
            {
                setAvarageFitness(0.0);
            }
        }
    }

    void pauseEvolution()
    {
        final switch(evolState)
        {
            case(EvolutionState.Running):
            {
                evolutionTid.send(thisTid, EvolutionCommand.Pause);
                return;
            }
            case(EvolutionState.Paused):
            {
                evolutionTid.send(thisTid, EvolutionCommand.Pause);
                return;
            }
            case(EvolutionState.Stoped):
            {
                return;
            }
        }
    }

    private
```

```
{
    enum EvolutionState
    {
        Stoped,
        Running,
        Paused
    }

    enum EvolutionCommand
    {
        Pause,
        Resume,
        Stop
    }

    __gshared EvolutionState evolState;
    __gshared GraphCompiler compiler;
    Tid evolutionTid;

    static void evolutionThread(shared EvolutionWindow wndShared)
    {
        Thread.getThis().isDaemon(true);

        EvolutionWindow wnd = cast()wndShared;
        try
        {
            auto progressBar = cast(ProgressBar)wnd.builder.getObject("EvolutionProgressBar");
            assert(progressBar !is null);

            evolState = EvolutionState.Running;
            scope(exit) evolState = EvolutionState.Stoped;

            wnd.project.programType.registerTypes();

            bool exit = false;
            bool paused = false;
            Tid parent = receiveOnly!Tid();

            void listener()
            {
                receiveTimeout(dur!"msecs"(1),
                    (Tid sender, EvolutionCommand command)
                    {
                        final switch(command)
                        {
                            case(EvolutionCommand.Resume):
                            {
                                evolState = EvolutionState.Running;
                                paused = false;
                                break;
                            }
                            case(EvolutionCommand.Pause):
                            {
                                evolState = EvolutionState.Paused;
                                paused = true;
                                break;
                            }
                            case(EvolutionCommand.Stop):
                            {
                                exit = true;
                                paused = false;
                                evolState = EvolutionState.Paused;
                                break;
                            }
                        }
                    });
            }

            void updater(double percent)
            {
                listener();
                assert(progressBar !is null);

                threadsEnter();
                progressBar.setFraction(percent);
                threadsLeave();
            }
            auto updaterDelegate = toDelegate(&updater);

            bool whenExit()
            {
                return exit;
            }
            auto whenExitDelegate = toDelegate(&whenExit);

            bool pauser()
```

```
                {
                    return paused;
                }
                auto pauserDelegate = toDelegate(&pauser);

                while(!exit)
                {
                    compiler.envolveGeneration(whenExitDelegate, "saves"
                        , updaterDelegate, pauserDelegate);
                }
                updater(0.0);
            } catch(OwnerTerminated e)
            {

            } catch(Exception e)
            {
                wnd.logger.logError(e.toString);
            } catch(Throwable th)
            {
                wnd.logger.logError(th.toString);
            }
        }
    }
}
```

5 Пакет GUI.GENERIC

```
module gui.generic;

import gtk.Builder;
import gtk.ApplicationWindow;
import gtk.ImageMenuItem;
import gtk.FileChooserDialog;
import gtk.AboutDialog;
import gdk.Pixbuf;
import gtk.Main;
import dlogg.log;

import project;
import application;

import std.file;

abstract class GenericWindow
{
    enum defaultWindowTittle = "Graph-isomorph";

    this(Application app, Builder builder, shared ILogger logger, Project project
        ,ApplicationWindow window)
    {
        this.app = app;
        mBuilder = builder;
        mLogger = logger;
        mProject = project;
        mWindow = window;

        window.setIconFromFile("icon_small.png");
        updateTittle();
    }

    Application application()
    {
        return app;
    }

    void updateTittle()
    {
        window.setTitle(defaultWindowTittle ~ " " ~ project.filename);
    }

    Builder builder()
    {
        return mBuilder;
    }

    shared(ILogger) logger()
    {
        return mLogger;
    }

    Project project()
    {
```

```
    return mProject;
}

ApplicationWindow window()
{
    return mWindow;
}

void updateContent()
{
    updateTittle();
}

void initProjectSaveLoad(string distinct)
{
    logger.logInfo("New project button setup");
    auto newItem = cast(ImageMenuItem)builder.getObject("NewProjectMenuItem"~distinct);
    assert(newItem !is null);
    newItem.addOnActivate((i)
    {
        try
        {
            auto dlg = new FileChooserDialog("Choose new project file"
                , window
                , FileChooserAction.SAVE
                , ["OK", "Cancel"]
                , [ResponseType.OK, ResponseType.CANCEL]
            );

            dlg.setDoOverwriteConfirmation(true);
            dlg.setCurrentFolder(getcwd);
            dlg.setCurrentName(Project.defaultProjectPath);

            switch(dlg.run)
            {
                case(ResponseType.OK):
                {
                    string filename = dlg.getFilename;
                    if(filename !is null)
                    {
                        project.recreate(filename);
                        application.updateAll();
                    }
                    dlg.destroy;
                    return;
                }
                default:
                {
                    dlg.destroy;
                    return;
                }
            }
        }
        catch(Throwable e)
        {
            logger.logError(e.toString);
        }
    });

    logger.logInfo("Open project button setup");
    auto openItem = cast(ImageMenuItem)builder.getObject("OpenProjectMenuItem"~distinct);
    assert(openItem !is null);
    openItem.addOnActivate((i)
    {
        try
        {
            auto dlg = new FileChooserDialogВыберите(" файлпроекта "
                , window
                , FileChooserAction.OPEN
                , ["OK", Отмена""]
                , [ResponseType.OK, ResponseType.CANCEL]
            );

            dlg.setCurrentFolder(getcwd);

            switch(dlg.run)
            {
                case(ResponseType.OK):
                {
                    string filename = dlg.getFilename;
                    if(filename !is null)
                    {
                        project.open(filename);
                        app.updateAll();
                    }
                    dlg.destroy;
                    return;
                }
                default:
                {
                    dlg.destroy;
                    return;
                }
            }
        }
        catch(Throwable e)
        {
            logger.logError(e.toString);
        }
    });

    logger.logInfo("Save as project button setup");
    auto saveAsItem = cast(ImageMenuItem)builder.getObject("SaveAsProjectMenuItem"~distinct);
    assert(saveAsItem !is null);
    saveAsItem.addOnActivate((i)
    {
        try
        {
            auto dlg = new FileChooserDialogВыберите(" файлпроекта "
                , window
                , FileChooserAction.SAVE
                , ["OK", Отмена""]
                , [ResponseType.OK, ResponseType.CANCEL]
            );

            dlg.setDoOverwriteConfirmation(true);
            dlg.setCurrentFolder(getcwd);
            dlg.setCurrentName(Project.defaultProjectPath);

            switch(dlg.run)
            {
                case(ResponseType.OK):
                {
                    string filename = dlg.getFilename;
                    if(filename !is null)
                    {
                        project.save(filename);
                        app.updateAll();
                    }
                    dlg.destroy;
                    return;
                }
                default:
                {
                    dlg.destroy;
                    return;
                }
            }
        }
        catch(Throwable e)
        {
            logger.logError(e.toString);
        }
    });

    logger.logInfo("Save project button setup");
    auto saveItem = cast(ImageMenuItem)builder.getObject("SaveProjectMenuItem"~distinct);
    assert(saveItem !is null);
    saveItem.addOnActivate((i)
    {
        try
        {
            project.save(project.filename);
            app.updateAll();
        }
        catch(Throwable e)
        {
            logger.logError(e.toString);
        }
    });

    logger.logInfo("Exit application button setup");
    auto exitItem = cast(ImageMenuItem)builder.getObject("ExitMenuItem"~distinct);
    assert(exitItem !is null);
    exitItem.addOnActivate((i)
    {
        try
        {
            Main.quit();
        }
        catch(Throwable e)
        {
```

```
                logger.logError(e.toString);
            }
        });
    }

    void initAboutDialog(string distinct)
    {
        auto helpItem = cast(ImageMenuItem)builder.getObject("AboutMenuItem"~distinct);
        assert(helpItem);

        helpItem.addOnActivate((i)
                {
                        auto dlg = new AboutDialog;
                        dlg.setProgramNameАСОИ(" поискаалгоритмовраспознаванияизоморфизмаграфов   \n" Курсовое"
                            ↪ проектированиеМГТУим  . НЭБаумана..\n" Научный" руководитель:
                            ↪ ФилипповичЮрийНиколаевич  ");
                        dlg.setAuthorsГуща([" АнтонВалерьевич "]);
                        dlg.setLicenseType(GtkLicense.MIT_X11);
                        dlg.addOnResponse((r,d) => dlg.destroy);
                        auto logo = new Pixbuf("icon.png");
                        dlg.setLogo(logo);
                        dlg.showAll;
                });
    }

    private
    {
        Builder mBuilder;
        shared ILogger mLogger;
        Project mProject;
        ApplicationWindow mWindow;
        Application app;
    }
}
```

6 Пакет GUI.RESULTS

```
module gui.results;

import gtk.Builder;
import gtk.MenuItem;
import gtk.ApplicationWindow;
import gtk.TreeView;
import gtk.TreeIter;
import gtk.TextView;
import gtk.Image;
import gtk.ListStore;

import gui.util;
import gui.generic;

import dlogg.log;

import project;
import application;

import std.conv;
import std.file;
import std.path;
import std.stdio;
import std.process;

import devol.individ;

class ResultsWindow : GenericWindow
{
    this(Application app, Builder builder, shared ILogger logger
        , Project project
        , ApplicationWindow settingsWindow
        , ApplicationWindow evoluitionWindow
        , ApplicationWindow resultsWindow)
    {
        super(app, builder, logger, project, resultsWindow);

        resultsWindow.hide();
        resultsWindow.addOnHide( (w) => onWindowHideShow(AppWindow.Results, true) );
        resultsWindow.addOnShow( (w) => onWindowHideShow(AppWindow.Results, false) );
        resultsWindow.addOnDelete( (e, w) { resultsWindow.hide; return true; } );

        auto showSettingsWndItem = cast(MenuItem)builder.getObject("ShowSettingsWndItem3");
        if(showSettingsWndItem is null)
        {
            logger.logError("ResultsWnd: failed to get show settings wnd item!");
            assert(false);
        }
        showSettingsWndItem.addOnActivate( (w) => settingsWindow.showAll() );

        auto showEvolutionWndItem = cast(MenuItem)builder.getObject("ShowEvolutionWndItem3");
        if(showEvolutionWndItem is null)
        {
            logger.logError("ResultsWnd: failed to get show evolution wnd item!");
            assert(false);
        }
        showEvolutionWndItem.addOnActivate( (w) => evoluitionWindow.showAll() );

        initProjectSaveLoad("3");
        initAboutDialog("3");

        initPopulationView();
    }

    private void setImage(IndAbstract graph, string wname)
    {
        try
        {
            enum tempImageDir = "./images";
            if(!tempImageDir.exists)
            {
                mkdirRecurse(tempImageDir);
            }

            string dotFilename = buildPath(tempImageDir, wname~".dot");
            string imageFilename = buildPath(tempImageDir, wname~".png");

            auto file = File(dotFilename, "w");
            file.writeln(graph.genDot());
            file.close();

            shell(text("dot -Tpng ", dotFilename, " > ", imageFilename));

            programImage.setFromFile(imageFilename);
        }
        catch(Exception e)
        {
            logger.logError("Failed to load image from graph for "~wname);
            logger.logError(e.msg);
        }
    }

    private TreeView individsView;
    private ListStore individsViewModel;
    private TextView programView;
    private Image programImage;

    void updatePopulation()
    {
        individsViewModel.clear();
        programView.getBuffer().setText("");
        programImage.clear();

        if(project.population !is null)
        {
                foreach(i, ind; project.population)
                {
                        auto iter = new TreeIter();
                        individsViewModel.insert(iter, -1);

                        individsViewModel.setValue(iter, 0, ind.name);
                        individsViewModel.setValue(iter, 1, to!string(ind.fitness));
                        individsViewModel.setValue(iter, 2, cast(int)i);
                }
        }
    }

    override void updateContent()
    {
        super.updateContent();
        updatePopulation();
    }

    void initPopulationView()
    {
        individsView = cast(TreeView)builder.getObject("IndividsTreeView");
        assert(individsView !is null);

        individsViewModel = new ListStore([GType.STRING, GType.STRING, GType.INT]);
        individsView.setModel(individsViewModel);
        programView = cast(TextView)builder.getObject("ProgramTextView");
        assert(programView !is null);

        programImage = cast(Image)builder.getObject("ProgramImage");
        assert(programImage !is null);

        individsView.addOnCursorChanged((v)
        {
            auto model = individsView.getModel();
            assert(model !is null);

            auto iter = individsView.getSelectedIter();
            if(iter is null)
            {
                programImage.clear();
                programView.getBuffer().setText("");
            }
            else
            {
                auto individId = model.getValueInt(iter, 2);
                if(individId < 0 || individId >= project.population.length) return;

                auto individ = project.population[cast(size_t)individId];
                assert(individ !is null);

                programView.getBuffer().setText(individ.programString);
                setImage(individ, individ.name);
            }
        });
    }
}
```

7 Пакет GUI.SETTINGS

```
module gui.settings;

import gtk.Builder;
import gtk.MenuItem;
import gtk.ApplicationWindow;
import gtk.Entry;
import gtk.MessageDialog;
import gtk.Widget;
import gtk.TreeView;
import gtk.TextView;
import gtk.TextIter;
import gtk.ListStore;
import gtk.TreeIter;

import gdk.Event;
import gobject.Value;

import gui.util;
import gui.generic;

import dlogg.log;

import evol.progtype;
import devol.operator;
import devol.operatormng;

import project;
import application;

import std.string;

class SettingsWindow : GenericWindow
{
    this(Application app, Builder builder, shared ILogger logger
        , Project project
        , ApplicationWindow settingsWindow
        , ApplicationWindow evoluitionWindow
        , ApplicationWindow resultsWindow)
    {
        super(app, builder, logger, project, settingsWindow);

        settingsWindow.showAll();
        settingsWindow.addOnHide( (w) => onWindowHideShow(AppWindow.Settings, true) );
        settingsWindow.addOnShow( (w) => onWindowHideShow(AppWindow.Settings, false) );
        settingsWindow.addOnDelete( (e, w) { settingsWindow.hide; return true; } );

        auto showEvolutionWndItem = cast(MenuItem)builder.getObject("ShowEvolutionWndItem1");
        if(showEvolutionWndItem is null)
        {
            logger.logError("SettingsWnd: failed to get show evolution wnd item!");
            assert(false);
        }
        showEvolutionWndItem.addOnActivate( (w) => evoluitionWindow.showAll() );

        auto showResultsWndItem = cast(MenuItem)builder.getObject("ShowResultsWndItem1");
        if(showResultsWndItem is null)
        {
            logger.logError("SettingsWnd: failed to get show results wnd item!");
            assert(false);
        }
        showResultsWndItem.addOnActivate( (w) => resultsWindow.showAll() );

        initProgtypeEntries();
        initOperatorsView();
        initProjectSaveLoad("1");
        initAboutDialog("1");
    }

    override void updateContent()
    {
        super.updateContent();
        reloadSettings();
    }

    private
    {
        static char cupper(char c)
        {
            immutable source = "qwertyuiopasdfghjklzxcvbnm";
            immutable dist   = "QWERTYUIOPASDFGHJKLZXCVBNM";

            foreach(i, sc; source)
            {
                if(sc == c) return dist[i];
            }

            return c;
        }

        template genEntryGetter(tt...)
        {
            enum field = tt[0];
            enum genEntryGetter = "auto "~field
                ~`Entry = cast(Entry)builder.getObject("`
                ~[cupper(cast(char)field[0])]~field[1..$]~`Entry");`"\n"
                ~`assert(`~field~`Entry !is null);`;
        }

        template genInitialSetupText(tts...)
        {
            enum field = tts[0];
            enum genInitialSetupText = field~`Entry.setText(project.programType.`~field~`.to!string);`;
        }
    }

    void initProgtypeEntries()
    {
        mixin(genEntryGetter!"progMinSize");
        mixin(genEntryGetter!"progMaxSize");
        mixin(genEntryGetter!"scopeMinSize");
        mixin(genEntryGetter!"scopeMaxSize");
        mixin(genEntryGetter!"newOpGenChance");
        mixin(genEntryGetter!"newScopeGenChance");
        mixin(genEntryGetter!"newLeafGenChance");
        mixin(genEntryGetter!"mutationChangeChance");
        mixin(genEntryGetter!"mutationReplaceChance");
        mixin(genEntryGetter!"mutationDeleteChance");
        mixin(genEntryGetter!"mutationAddLineChance");
        mixin(genEntryGetter!"mutationRemoveLineChance");
        mixin(genEntryGetter!"maxMutationChange");
        mixin(genEntryGetter!"mutationChance");
        mixin(genEntryGetter!"crossingoverChance");
        mixin(genEntryGetter!"copyingPart");
        mixin(genEntryGetter!"deleteMutationRiseGenomeSize");
        mixin(genEntryGetter!"maxGenomeSize");
        mixin(genEntryGetter!"populationSize");
        mixin(genEntryGetter!"graphPermuteChance");
        mixin(genEntryGetter!"graphNodesCountMin");
        mixin(genEntryGetter!"graphNodesCountMax");
        mixin(genEntryGetter!"graphLinksCountMin");
        mixin(genEntryGetter!"graphLinksCountMax");
        mixin(genEntryGetter!"graphPermutesCountMin");
        mixin(genEntryGetter!"graphPermutesCountMax");
```

```
void showInvalidValueDialog(T)(string value)
{
    auto dialog = new MessageDialog(window
        , GtkDialogFlags.MODAL
        , GtkMessageType.ERROR
        , GtkButtonsType.CLOSE
        , "%s"
        , text("Expected value of type ", T.stringof, ", but got '", value,"'!"));
    dialog.addOnResponse( (r, d) => dialog.destroy );
    dialog.run();
}

bool delegate(Event, Widget) tryFillValue(T, string field)(Entry entry)
{
    return (e, w)
    {
        scope(failure)
        {
            showInvalidValueDialog!T(entry.getText);
            return false;
        }
        mixin("project.programType."~field~" = entry.getText.to!T;");
        return false;
    };
}

template genFocusSignal(tts...)
{
    enum field = tts[0];
    mixin(`alias T = typeof(project.programType.`~field~`);`);
    enum genFocusSignal = field~"Entry.addOnFocusOut(tryFillValue!("~T.stringof~`,"`~field~`")(`~field~`
        ↪ Entry));`;
}

mixin(genFocusSignal!"progMinSize");
mixin(genFocusSignal!"progMaxSize");
mixin(genFocusSignal!"scopeMinSize");
mixin(genFocusSignal!"scopeMaxSize");
mixin(genFocusSignal!"newOpGenChance");
mixin(genFocusSignal!"newScopeGenChance");
mixin(genFocusSignal!"newLeafGenChance");
mixin(genFocusSignal!"mutationChangeChance");
mixin(genFocusSignal!"mutationReplaceChance");
mixin(genFocusSignal!"mutationDeleteChance");
mixin(genFocusSignal!"mutationAddLineChance");
mixin(genFocusSignal!"mutationRemoveLineChance");
mixin(genFocusSignal!"maxMutationChange");
mixin(genFocusSignal!"mutationChance");
mixin(genFocusSignal!"crossingoverChance");
mixin(genFocusSignal!"copyingPart");
mixin(genFocusSignal!"deleteMutationRiseGenomeSize");
mixin(genFocusSignal!"maxGenomeSize");
mixin(genFocusSignal!"populationSize");
mixin(genFocusSignal!"graphPermuteChance");
mixin(genFocusSignal!"graphNodesCountMin");
mixin(genFocusSignal!"graphNodesCountMax");
mixin(genFocusSignal!"graphLinksCountMin");
mixin(genFocusSignal!"graphLinksCountMax");
mixin(genFocusSignal!"graphPermutesCountMin");
mixin(genFocusSignal!"graphPermutesCountMax");

mixin(genInitialSetupText!"progMinSize");
mixin(genInitialSetupText!"progMaxSize");
mixin(genInitialSetupText!"scopeMinSize");
mixin(genInitialSetupText!"scopeMaxSize");
mixin(genInitialSetupText!"newOpGenChance");
mixin(genInitialSetupText!"newScopeGenChance");
mixin(genInitialSetupText!"newLeafGenChance");
mixin(genInitialSetupText!"mutationChangeChance");
mixin(genInitialSetupText!"mutationReplaceChance");
mixin(genInitialSetupText!"mutationDeleteChance");
mixin(genInitialSetupText!"mutationAddLineChance");
mixin(genInitialSetupText!"mutationRemoveLineChance");
mixin(genInitialSetupText!"maxMutationChange");
mixin(genInitialSetupText!"mutationChance");
mixin(genInitialSetupText!"crossingoverChance");
mixin(genInitialSetupText!"copyingPart");
mixin(genInitialSetupText!"deleteMutationRiseGenomeSize");
mixin(genInitialSetupText!"maxGenomeSize");
mixin(genInitialSetupText!"populationSize");
mixin(genInitialSetupText!"graphPermuteChance");
mixin(genInitialSetupText!"graphNodesCountMin");
mixin(genInitialSetupText!"graphNodesCountMax");
mixin(genInitialSetupText!"graphLinksCountMin");
mixin(genInitialSetupText!"graphLinksCountMax");
mixin(genInitialSetupText!"graphPermutesCountMin");
```

```
    mixin(genInitialSetupText!"graphPermutesCountMax");
}

void reloadSettings()
{
    mixin(genEntryGetter!"progMinSize");
    mixin(genEntryGetter!"progMaxSize");
    mixin(genEntryGetter!"scopeMinSize");
    mixin(genEntryGetter!"scopeMaxSize");
    mixin(genEntryGetter!"newOpGenChance");
    mixin(genEntryGetter!"newScopeGenChance");
    mixin(genEntryGetter!"newLeafGenChance");
    mixin(genEntryGetter!"mutationChangeChance");
    mixin(genEntryGetter!"mutationReplaceChance");
    mixin(genEntryGetter!"mutationDeleteChance");
    mixin(genEntryGetter!"mutationAddLineChance");
    mixin(genEntryGetter!"mutationRemoveLineChance");
    mixin(genEntryGetter!"maxMutationChange");
    mixin(genEntryGetter!"mutationChance");
    mixin(genEntryGetter!"crossingoverChance");
    mixin(genEntryGetter!"copyingPart");
    mixin(genEntryGetter!"deleteMutationRiseGenomeSize");
    mixin(genEntryGetter!"maxGenomeSize");
    mixin(genEntryGetter!"populationSize");
    mixin(genEntryGetter!"graphPermuteChance");
    mixin(genEntryGetter!"graphNodesCountMin");
    mixin(genEntryGetter!"graphNodesCountMax");
    mixin(genEntryGetter!"graphLinksCountMin");
    mixin(genEntryGetter!"graphLinksCountMax");
    mixin(genEntryGetter!"graphPermutesCountMin");
    mixin(genEntryGetter!"graphPermutesCountMax");

    mixin(genInitialSetupText!"progMinSize");
    mixin(genInitialSetupText!"progMaxSize");
    mixin(genInitialSetupText!"scopeMinSize");
    mixin(genInitialSetupText!"scopeMaxSize");
    mixin(genInitialSetupText!"newOpGenChance");
    mixin(genInitialSetupText!"newScopeGenChance");
    mixin(genInitialSetupText!"newLeafGenChance");
    mixin(genInitialSetupText!"mutationChangeChance");
    mixin(genInitialSetupText!"mutationReplaceChance");
    mixin(genInitialSetupText!"mutationDeleteChance");
    mixin(genInitialSetupText!"mutationAddLineChance");
    mixin(genInitialSetupText!"mutationRemoveLineChance");
    mixin(genInitialSetupText!"maxMutationChange");
    mixin(genInitialSetupText!"mutationChance");
    mixin(genInitialSetupText!"crossingoverChance");
    mixin(genInitialSetupText!"copyingPart");
    mixin(genInitialSetupText!"deleteMutationRiseGenomeSize");
    mixin(genInitialSetupText!"maxGenomeSize");
    mixin(genInitialSetupText!"populationSize");
    mixin(genInitialSetupText!"graphPermuteChance");
    mixin(genInitialSetupText!"graphNodesCountMin");
    mixin(genInitialSetupText!"graphNodesCountMax");
    mixin(genInitialSetupText!"graphLinksCountMin");
    mixin(genInitialSetupText!"graphLinksCountMax");
    mixin(genInitialSetupText!"graphPermutesCountMin");
    mixin(genInitialSetupText!"graphPermutesCountMax");
}

void initOperatorsView()
{
    auto operatorsView = cast(TreeView)builder.getObject("OperatorsView");
    auto operatorNameEntry = cast(Entry)builder.getObject("OperatorNameEntry");
    auto operatorDescriptionView = cast(TextView)builder.getObject("OperatorDescriptionView");

    auto model = new ListStore([GType.STRING]);
    operatorsView.setModel(model);

    operatorsView.addOnCursorChanged((v)
    {
        auto opmng = OperatorMng.getSingleton();
        auto model = operatorsView.getModel();
        assert(model !is null);

        auto iter = operatorsView.getSelectedIter();
        if(iter is null)
        {
            operatorNameEntry.setText("");
            operatorDescriptionView.getBuffer().setText("");
        }
        else
        {
            auto value = model.getValue(iter, 0);
            auto opName = value.getString();
            auto operator = opmng.getOperator(opName);
```

```
                operatorNameEntry.setText(operator.name);
                operatorDescriptionView.getBuffer().setText(operator.disrc);
            }
        });

        auto opmng = OperatorMng.getSingleton();
        foreach(operator; OperatorMng.getSingleton)
        {
            auto iter = new TreeIter;
            model.insert(iter, -1);
            model.setValue(iter, 0, operator.name);
        }
    }
}
```

8 Пакет GUI.UTIL

```
module gui.util;

import gtk.ApplicationWindow;
import gtk.Main;
import gtk.TextView;
import gtk.TextIter;

enum AppWindow
{
    Settings,
    Evolution,
    Results
}

void onWindowHideShow(AppWindow type, bool isClosed)
{
    static bool settingsClosed = false;
    static bool evolutionClosed = true;
    static bool resultsClosed = true;

    final switch(type)
    {
        case(AppWindow.Settings): settingsClosed = isClosed; break;
        case(AppWindow.Evolution): evolutionClosed = isClosed; break;
        case(AppWindow.Results): resultsClosed  = isClosed; break;
    }

    if(settingsClosed && evolutionClosed && resultsClosed)
    {
        Main.quit();
    }
}
```

9 Пакет GRAPH.CONNECTIVITY

```
module graph.connectivity;

import graph.directed;
import std.algorithm;
import std.range;
import std.array;
import std.conv;

/**
*   Graph implemented via connectivity lists.
*/
class ConnListGraph : IDirectedGraph
{
    /**
    *   Loading graph from raw data.
    */
    void load(InputRange!Edge input)
    {
        foreach(edge; input)
        {
            if(edge.source !in lists)
            {
                Weight[Node] empty;
                lists[edge.source] = empty;
            }
            auto list = lists[edge.source];

            if(edge.dist in list)
            {
                assert(edge.weight == list[edge.dist]);
            }
            list[edge.dist] = edge.weight;
            lists[edge.source] = list;
        }
    }

    /**
    *   Returns graph nodes set
    */
    InputRange!string nodes()
    {
        bool[Node] nodes;

        foreach(source, list; lists)
        {
            nodes[source] = true;
            foreach(dist, weight; list)
            {
                nodes[dist] = true;
            }
        }

        return nodes.keys.inputRangeObject;
    }

    /**
    *   Returns graph weights set
    */
    InputRange!string weights()
    {
        bool[Weight] weights;

        foreach(source, list; lists)
        {
            foreach(dist, weight; list)
            {
                weights[weight] = true;
            }
        }
        return weights.keys.inputRangeObject;
    }

    /**
    *   Returns graph edges set
    */
    InputRange!Edge edges()
    {
        auto builder = appender!(Edge[]);

        foreach(source, list; lists)
        {
            foreach(dist, weight; list)
            {
                builder.put(Edge(source, dist, weight));
            }
        }

        return builder.data.inputRangeObject;
    }

    /**
    *   Returns indexed graph edges set
    */
    InputRange!IndexedEdge indexedEdges()
    {
        auto builder = appender!(IndexedEdge[]);
        auto nodesArr = nodes.array;

        foreach(edge; edges)
        {
            builder.put(IndexedEdge(
                    nodesArr.countUntil(edge.source),
                    nodesArr.countUntil(edge.dist) ));
        }

        return builder.data.inputRangeObject;
    }

    string genDot()
    {
        Weight[Node] getFirst(out Node node)
```

```
{
    foreach(k, list; lists)
    {
        node = k;
        return list;
    }
    assert(false);
}

string genNodeName(size_t i)
{
    if(i >= 1000)
    {
        return text("n",i);
    } else if(i >= 100)
    {
        return text("n0",i);
    } else if(i >= 10)
    {
        return text("n00",i);
    } else
    {
        return text("n00",i);
    }
}

string genForList(ref size_t i, Node node, Weight[Node] list, ref size_t[Node] nodeMap)
{
    auto builder = appender!string;

    string nodeName;
    if(node !in nodeMap)
    {
        nodeName = genNodeName(i);
        nodeMap[node] = i;
        i+=1;

        builder.put(nodeName);
        builder.put(" ;\n");

        builder.put(nodeName);
        builder.put(`[label="`);
        builder.put(node.to!string);
        builder.put(`"] ;`"\n");
    } else
    {
        nodeName = genNodeName(nodeMap[node]);
    }

    foreach(dist, weight; list)
    {
        bool genLabel = false;
        if(dist !in nodeMap)
        {
            nodeMap[dist] = i;
            i+=1;
            genLabel = true;
        }
        builder.put(nodeName);
        builder.put(" -> ");
        builder.put(genNodeName(nodeMap[dist]));
        builder.put(` [ label = "`);
        builder.put(weight.to!string);
        builder.put(`" ] ;`);
        builder.put("\n");

        if(genLabel)
        {
            builder.put(genNodeName(nodeMap[dist]));
            builder.put(`[label="`);
            builder.put(dist.to!string);
            builder.put(`"] ;`"\n");
        }
    }
    return builder.data;
}

auto builder = appender!string;

builder.put(`digraph "" {`"\n");
size_t[Node] nodeMap;

size_t i = 0;
foreach(node, list; lists)
{
    builder.put(genForList(i, node, list, nodeMap));
```

```
        }

        builder.put('}'"\n");

        return builder.data;
    }

    private
    {
        Weight[Node][Node] lists;
    }
}

unittest
{
    import std.algorithm;

    auto edges = [
        IDirectedGraph.Edge("a", "b", "1"),
        IDirectedGraph.Edge("b", "a", "2")
    ];

    auto graph = new ConnListGraph();
    graph.load(edges.inputRangeObject);

    assert(graph.nodes.array.sort.equal(["a", "b"]));
    assert(graph.weights.array.sort.equal(["1", "2"]));
}
unittest
{
    import std.process;
    import std.stdio;

    auto edges = [
        IDirectedGraph.Edge("a", "b", "1"),
        IDirectedGraph.Edge("b", "a", "2"),
        IDirectedGraph.Edge("c", "a", "3"),
        IDirectedGraph.Edge("c", "b", "1")
    ];

    auto graph = new ConnListGraph();
    graph.load(edges.inputRangeObject);

    auto file = File("test.dot", "w");
    file.writeln(graph.genDot);
    file.close();

    shell("dot -Tpng test.dot > test.png");
    shell("gwenview test.png");
}
```

10 Пакет GRAPH.DIRECTED

```
module graph.directed;

import std.range;
import std.conv;

/**
*   Generic interface for directed graph. Nodes are marked with
*   strings, edges are marked with strings too.
*/
interface IDirectedGraph
{
    /**
    *   Unpacked graph edge.
    */
    struct Edge
    {
        /// Edge source
        string source;
        /// Edge dist
        string dist;
        /// Edge weight
        string weight;

        string toString()
        {
            if(weight == "")
            {
                return source ~ " -> " ~ dist;
            }
            else
            {
                return source ~ "- " ~ weight ~ " -> " ~ dist;
            }
        }
    }

    /**
    *   Operating with indexes is more handy for genetic programs.
    */
    struct IndexedEdge
    {
        size_t source;
        size_t dist;

        string toString()
        {
            return text(source, " -> ", dist);
        }
    }

    alias string Node;
    alias string Weight;

    /**
    *   Loading graph from raw data.
    */
    void load(InputRange!Edge input);

    /**
    *   Returns graph nodes set
    */
    InputRange!Node nodes();

    /**
    *   Returns graph weights set
    */
    InputRange!Weight weights();

    /**
    *   Returns graph edges set
    */
    InputRange!Edge edges();

    /**
    *   Returns indexed graph edges set
    */
    InputRange!IndexedEdge indexedEdges();

    /**
    *   Generates dot description for
    *   visualization.
    */
    string genDot();
}
```

11 Пакет EVOL.COMPILER

```
module evol.compiler;

import devol.compiler;
import devol.population;

import evol.individ;
import evol.world;
import evol.progtype;

import project;

alias Population!( getDefChars, GraphIndivid ) GraphPopulation;

class GraphCompilation : GameCompilation
{
    Project project;

        this(Project project, void delegate() updateGenerationInfo)
        {
            this.project = project;
                this.updateGenerationInfo = updateGenerationInfo;
        }

    bool stopCond(ref int step, IndAbstract ind, WorldAbstract world)
    {
        return step >= 1;
    }

    void drawStep(IndAbstract ind, WorldAbstract world)
    {

    }

    void drawFinal(PopAbstract pop, WorldAbstract world)
    {
        project.population = cast(GraphPopulation)pop;
        updateGenerationInfo();
    }

    int roundsPerInd()
    {
        return 10;
    }

    private void delegate() updateGenerationInfo;
}


alias GraphCompiler = Compiler!(
      GraphCompilation
    , Evolutor
    , ProgramType
    , GraphPopulation
    , GraphWorld);
```

12 Пакет EVOL.INDIVID

```
module evol.individ;

import devol.individ;
import devol.argument;
import devol.std.argvoid;

import dyaml.all;

import std.container;

import evol.types.argedge;
import evol.world;
import devol.std.argpod;

class GraphIndivid : Individ
{
    this()
    {

    }

    this(Individ ind)
    {
        this();
        loadFrom(ind);
    }

    override void initialize(WorldAbstract aworld)
    {
        super.initialize(aworld);

        GraphWorld world = cast(GraphWorld)aworld;
        assert(world);

        mStack.mStack.clear;

        mFirstGraph.mStack.clear;
        foreach(edge; world.firstGraph.indexedEdges)
        {
            mFirstGraph.stackPush(new ArgEdge(edge));
        }

        mSecondGraph.mStack.clear;
        foreach(edge; world.secondGraph.indexedEdges)
        {
            mSecondGraph.stackPush(new ArgEdge(edge));
        }
    }
```

```
override @property GraphIndivid dup()
{
    auto ind = new GraphIndivid();
    ind.mFitness = mFitness;

    ind.mProgram = [];
    foreach(line; mProgram)
        ind.mProgram ~= line.dup;

    ind.mMemory = [];
    foreach(line; mMemory)
        ind.mMemory ~= line.dup;

    ind.inVals = [];
    foreach(line; inVals)
        ind.inVals ~= line.dup;

    ind.outVals = [];
    foreach(line; outVals)
        ind.outVals ~= line.dup;

    return ind;
}

static GraphIndivid loadYaml(Node node)
{
    auto ind = Individ.loadYaml(node);
    auto ant = new GraphIndivid();
    ant.mFitness = ind.fitness;

    foreach(line; ind.program)
        ant.mProgram ~= line.dup;
    foreach(line; ind.memory)
        ant.mMemory ~= line.dup;
    foreach(line; ind.invals)
        ant.inVals ~= line.dup;
    foreach(line; ind.outvals)
        ant.outVals ~= line.dup;

    return ant;
}

struct Stack(T)
{
    void stackPush(T arg)
    {
        mStack.insertFront = arg;
    }

    T stackPop()
    {
        if(mStack.empty)
        {
            return new T;
        }
        else
        {
            auto arg = mStack.front;
            mStack.removeFront();
            return arg;
        }
    }

    void stackSwap()
    {
        if(mStack.empty) return;

        auto a1 = mStack.front;
        mStack.removeFront;

        if(mStack.empty)
        {
            mStack.insertFront = a1;
        } else
        {
            auto a2 = mStack.front;
            mStack.removeFront;

            mStack.insertFront = a1;
            mStack.insertFront = a2;
        }
    }

    void stackDup()
    {
        if(mStack.empty) return;
        mStack.insertFront = mStack.front;
    }

    void stackOver()
    {
        if(mStack.empty) return;

        auto a1 = mStack.front;
        mStack.removeFront;

        if(mStack.empty)
        {
            mStack.insertFront = a1;
        } else
        {
            auto a2 = mStack.front;
            mStack.removeFront;

            mStack.insertFront = a2;
            mStack.insertFront = a1;
            mStack.insertFront = a2;
        }
    }

    void stackRot()
    {
        if(mStack.empty) return;

        auto a1 = mStack.front;
        mStack.removeFront;

        if(mStack.empty)
        {
            mStack.insertFront = a1;
        } else
        {
            auto a2 = mStack.front;
            mStack.removeFront;

            if(mStack.empty)
            {
                mStack.insertFront = a2;
                mStack.insertFront = a1;
            } else
            {
                auto a3 = mStack.front;
                mStack.removeFront;

                mStack.insertFront = a2;
                mStack.insertFront = a1;
                mStack.insertFront = a3;
            }
        }
    }

    void stackDrop()
    {
        if(mStack.empty) return;

        mStack.removeFront;
    }

    DList!T mStack;
}

ref Stack!(ArgPod!double) genericStack()
{
    return mStack;
}

ref Stack!ArgEdge firstGraphStack()
{
    return mFirstGraph;
}

ref Stack!ArgEdge secondGraphStack()
{
    return mSecondGraph;
}

void answer(bool value)
{
    mAnswer = value;
}
```

```
    bool answer()
    {
        return mAnswer;
    }

    private
    {
        bool mAnswer = false;
        Stack!(ArgPod!double) mStack;
        Stack!ArgEdge mFirstGraph;
        Stack!ArgEdge mSecondGraph;
    }
}
```

13 Пакет EVOL.PROGTYPE

```
module evol.progtype;

import devol.programtype;
import devol.typemng;
import devol.operatormng;

import devol.std.typepod;
import std.conv;
import std.range;
import std.math;

import evol.operators.and;
import evol.operators.not;
import evol.operators.opif;
import evol.operators.opwhile;
import evol.operators.or;
import evol.operators.plus;
import evol.operators.mult;
import evol.operators.div;
import evol.operators.relation;
import evol.operators.gpop;
import evol.operators.gpush;
import evol.operators.gdup;
import evol.operators.gover;
import evol.operators.grot;
import evol.operators.gswap;
import evol.operators.ipop;
import evol.operators.ipush;
import evol.operators.idup;
import evol.operators.iover;
import evol.operators.irot;
import evol.operators.iswap;
import evol.operators.construct;
import evol.operators.dist;
import evol.operators.source;
import evol.operators.idcast;
import evol.operators.round;
import evol.operators.answer;

import evol.types.typeedge;
import evol.individ;
import evol.world;

import std.algorithm;
import std.range;

class ProgramType : ProgTypeAbstract
{
    this()
    {
        registerTypes();
    }

    void registerTypes()
    {
        auto tmng = TypeMng.getSingleton();
        auto omng = OperatorMng.getSingleton();

        auto types = tmng.strings;
        if(types.find("Typebool").empty)
        {
            tmng.registerType!TypeBool();
        }
        if(types.find("Typeint").empty)
        {
            tmng.registerType!TypeInt();
    }
    if(types.find("Typedouble").empty)
    {
        tmng.registerType!TypeDouble();
    }
    if(types.find("TypeEdge").empty)
    {
        tmng.registerType!TypeEdge();
    }

    auto ops = omng.strings;
    void registerOperator(T)(string name)
    {
        assert(name != "");
        if(ops.find(name).empty)
        {
            omng.registerOperator!T();
        }
    }

    registerOperator!IfOperator("if");
    registerOperator!WhileOperator("while");
    registerOperator!AndOperator("&&");
    registerOperator!OrOperator("||");
    registerOperator!NotOperator("!");

    registerOperator!PlusOperator("+");
    registerOperator!MultOperator("*");
    registerOperator!DivOperator("/");

    registerOperator!IntEqualOperator("== (int)");
    registerOperator!IntGreaterOperator("> (int)");
    registerOperator!IntLesserOperator("< (int)");
    registerOperator!IntGreaterEqualOperator(">= (int)");
    registerOperator!IntLesserEqualOperator("<= (int)");

    registerOperator!DoubleEqualOperator("== (double)");
    registerOperator!DoubleGreaterOperator("> (double)");
    registerOperator!DoubleLesserOperator("< (double)");

    registerOperator!GenericPopOperator("gpop");
    registerOperator!GenericPushOperator("gpush");
    registerOperator!GenericDupOperator("gdup");
    registerOperator!GenericOverOperator("gover");
    registerOperator!GenericRotOperator("grot");
    registerOperator!GenericSwapOperator("gswap");

    registerOperator!InputPopFirstOperator("ipop1");
    registerOperator!InputPushFirstOperator("ipush1");
    registerOperator!InputDupFirstOperator("idup1");
    registerOperator!InputOverFirstOperator("iover1");
    registerOperator!InputRotFirstOperator("irot1");
    registerOperator!InputSwapFirstOperator("iswap1");

    registerOperator!InputPopSecondOperator("ipop2");
    registerOperator!InputPushSecondOperator("ipush2");
    registerOperator!InputDupSecondOperator("idup2");
    registerOperator!InputOverSecondOperator("iover2");
    registerOperator!InputRotSecondOperator("irot2");
    registerOperator!InputSwapSecondOperator("iswap2");

    registerOperator!ConstructOperator("construct");
    registerOperator!GetSourceOperator("getSource");
    registerOperator!GetDistOperator("getDist");

    registerOperator!IntDoubleCastOperator("cast");
    registerOperator!RoundOperator("round");
    registerOperator!AnswerOperator("answer");
}

private uint mProgMinSize = 4;
@property uint progMinSize()
{
    return mProgMinSize;
}

@property void progMinSize(uint val)
{
    mProgMinSize = val;
}

private uint mProgMaxSize = 8;
@property uint progMaxSize()
{
    return mProgMaxSize;
}
```

```
@property void progMaxSize(uint val)
{
    mProgMaxSize = val;
}

private float mNewOpGenChacne = 0.3;
@property float newOpGenChance()
{
    return mNewOpGenChacne;
}

@property void newOpGenChance(float val)
{
    mNewOpGenChacne = val;
}

private float mNewScopeGenChance = 0.1;
@property float newScopeGenChance()
{
    return mNewScopeGenChance;
}

@property void newScopeGenChance(float val)
{
    mNewScopeGenChance = val;
}

private float mNewLeafGenChance = 0.6;
@property float newLeafGenChance()
{
    return mNewLeafGenChance;
}

@property void newLeafGenChance(float val)
{
    mNewLeafGenChance = val;
}

private uint mScopeMinSize = 2;
@property uint scopeMinSize()
{
    return mScopeMinSize;
}

@property void scopeMinSize(uint val)
{
    mScopeMinSize = val;
}

private uint mScopeMaxSize = 5;
@property uint scopeMaxSize()
{
    return mScopeMaxSize;
}

@property void scopeMaxSize(uint val)
{
    mScopeMaxSize = val;
}

private float mMutationChance = 0.3;
@property float mutationChance()
{
    return mMutationChance;
}

@property void mutationChance(float val)
{
    mMutationChance = val;
}

private float mCrossingoverChance = 0.7;
@property float crossingoverChance()
{
    return mCrossingoverChance;
}

@property void crossingoverChance(float val)
{
    mCrossingoverChance = val;
}

private float mMutationChangeChance = 0.5;
@property float mutationChangeChance()
{
    return mMutationChangeChance;
}

@property void mutationChangeChance(float val)
{
    mMutationChangeChance = val;
}

private float mMutationReplaceChance = 0.3;
@property float mutationReplaceChance()
{
    return mMutationReplaceChance;
}

@property void mutationReplaceChance(float val)
{
    mMutationReplaceChance = val;
}

private float mMutationDeleteChance = 0.2;
@property float mutationDeleteChance()
{
    return mMutationDeleteChance;
}

@property void mutationDeleteChance(float val)
{
    mMutationDeleteChance = val;
}

private float mMutationAddLineChance = 0.1;
@property float mutationAddLineChance()
{
    return mMutationAddLineChance;
}

@property void mutationAddLineChance(float val)
{
    mMutationAddLineChance = val;
}

private float mMutationRemoveLineChance = 0.05;
@property float mutationRemoveLineChance()
{
    return mMutationRemoveLineChance;
}

@property void mutationRemoveLineChance(float val)
{
    mMutationRemoveLineChance = val;
}

private string mMaxMutationChange = "100";
@property string maxMutationChange()
{
    return mMaxMutationChange;
}

@property void maxMutationChange(string val)
{
    mMaxMutationChange = val;
}

private float mCopyingPart = 0.1;
@property float copyingPart()
{
    return mCopyingPart;
}

@property void copyingPart(float val)
{
    mCopyingPart = val;
}

private size_t mDeleteMutationRiseGenomeSize = 200;
@property size_t deleteMutationRiseGenomeSize()
{
    return mDeleteMutationRiseGenomeSize;
}

@property void deleteMutationRiseGenomeSize(size_t val)
{
    mDeleteMutationRiseGenomeSize = val;
}

private size_t mMaxGenomeSize = 300;
```

```
@property size_t maxGenomeSize()
{
    return mMaxGenomeSize;
}

@property void maxGenomeSize(size_t val)
{
    mMaxGenomeSize = val;
}

private size_t mPopulationSize = 10;
@property size_t populationSize()
{
    return mPopulationSize;
}

@property void populationSize(size_t val)
{
    mPopulationSize = val;
}

private double mGraphPermuteChance = 0.5;
@property double graphPermuteChance()
{
    return mGraphPermuteChance;
}

@property void graphPermuteChance(double val)
in
{
    assert(0.0 <= val && val <= 1.1, "Not a chance!");
}
body
{
    mGraphPermuteChance = val;
}

private size_t mGraphNodesCountMin = 3;
@property size_t graphNodesCountMin()
{
    return mGraphNodesCountMin;
}

@property void graphNodesCountMin(size_t val)
in
{
    assert(val <= mGraphNodesCountMax, "Must be <= graphNodesCountMax");
}
body
{
    mGraphNodesCountMin = val;
}

private size_t mGraphNodesCountMax = 10;
@property size_t graphNodesCountMax()
{
    return mGraphNodesCountMax;
}

@property void graphNodesCountMax(size_t val)
in
{
    assert(val >= mGraphNodesCountMin, "Must be >= graphNodesCountMin");
}
body
{
    mGraphNodesCountMax = val;
}

private size_t mGraphLinksCountMin = 3;
@property size_t graphLinksCountMin()
{
    return mGraphLinksCountMin;
}

@property void graphLinksCountMin(size_t val)
in
{
    assert(val <= mGraphLinksCountMax, "Must be <= graphLinksCountMax");
}
body
{
    mGraphLinksCountMin = val;
}

private size_t mGraphLinksCountMax = 6;
```

```
    @property size_t graphLinksCountMax()
    {
        return mGraphLinksCountMax;
    }

    @property void graphLinksCountMax(size_t val)
    in
    {
        assert(val >= mGraphLinksCountMin, "Must be >= graphLinksCountMin");
    }
    body
    {
        mGraphLinksCountMax = val;
    }


    private size_t mGraphPermutesCountMin = 2;
    @property size_t graphPermutesCountMin()
    {
        return mGraphPermutesCountMin;
    }

    @property void graphPermutesCountMin(size_t val)
    in
    {
        assert(val <= mGraphPermutesCountMax, "Must be <= graphPermutesCountMax");
    }
    body
    {
        mGraphPermutesCountMin = val;
    }

    private size_t mGraphPermutesCountMax = 4;
    @property size_t graphPermutesCountMax()
    {
        return mGraphPermutesCountMax;
    }

    @property void graphPermutesCountMax(size_t val)
    in
    {
        assert(val >= mGraphPermutesCountMin, "Must be >= graphPermutesCountMin");
    }
    body
    {
        mGraphPermutesCountMax = val;
    }

    Line[] initValues(WorldAbstract pWorld)
    {
        return new Line[0];
    }

    double getFitness(IndAbstract pInd, WorldAbstract pWorld, double time)
    {
        auto ind = cast(GraphIndivid)pInd;
        auto world = cast(GraphWorld)pWorld;
        assert(ind);
        assert(world);

        size_t n = world.firstGraph.nodes.walkLength + world.secondGraph.nodes.walkLength;
        enum tPerNode = 0.1;
        double ft = cast(double)(1.0 / (1.0 + exp( 5.0 / (tPerNode*n) * time - 5.0)));
        double fa = ind.answer == world.correctAnswer ? 1.0 : 0.0;

        if(fa < 0.9) ft = 0.0;

        return (ft + fa) / 2;
    }
}
```

14 Пакет EVOL.WORLD

```
module evol.world;

import devol.world;

import std.algorithm;
import std.array;
import std.random;
import std.range;
import std.file;
import std.stdio;
import std.path;
import std.process;

import graph.directed;
import graph.connectivity;

import evol.progtype;

class GraphWorld : WorldAbstract
{
    ProgramType programType;

    this()
    {
        assert(false);
    }

    this(ProgramType programType,
         void delegate(IDirectedGraph, IDirectedGraph) updateDrawDel)
    {
        this.programType = programType;
        this.updateDrawDel = updateDrawDel;
    }

    void initialize()
    {
        genUniqName(true);
        initInput();

        updateDrawDel(mInputGraphFirst, mInputGraphSecond);
    }

    IDirectedGraph firstGraph()
    {
        return mInputGraphFirst;
    }

    IDirectedGraph secondGraph()
    {
        return mInputGraphSecond;
    }

    bool correctAnswer()
    {
        return mAnswer;
    }

    private
    {
        void delegate(IDirectedGraph, IDirectedGraph) updateDrawDel;
        IDirectedGraph mInputGraphFirst;
        IDirectedGraph mInputGraphSecond;
        bool mAnswer;

        void initInput()
        {
            mInputGraphFirst = generateGraph(
                  uniform!"[]"(programType.graphNodesCountMin, programType.graphNodesCountMax)
                , uniform!"[]"(programType.graphLinksCountMin, programType.graphLinksCountMax));
            if(getChance(programType.graphPermuteChance))
            {
                mInputGraphSecond
                    = permuteGraph(mInputGraphFirst
                        , uniform!"[]"(programType.graphPermutesCountMin
                                     , programType.graphPermutesCountMax));
                mAnswer = true;
            } else
            {
                mInputGraphSecond
                    = generateGraph(
                  uniform!"[]"(programType.graphNodesCountMin, programType.graphNodesCountMax)
                , uniform!"[]"(programType.graphLinksCountMin, programType.graphLinksCountMax));

                mAnswer = false;
            }
        }

        static bool getChance(float val)
        {
            return uniform!"[]"(0.0,1.0) <= val;
        }

        static string genUniqName(bool clearMemory = false)
        {
            static bool[string] memory;
            if(clearMemory)
            {
                bool[string] clean;
                memory = clean;
                return "";
            }

            immutable alphabet = "qwertyuiopasdfghjklzxcvbnm";
            string genString(size_t l)
            {
                auto builder = appender!string;
                foreach(i; 0..l)
                    builder.put(alphabet[uniform(0,alphabet.length)]);
                return builder.data;
            }

            size_t i = 1;
            while(true)
            {
                string name = genString(i);
                if(name in memory)
                {
                    i++;
                } else
                {
                    return name;
                }
            }
        }

        static IDirectedGraph generateGraph(size_t nodesCount, size_t linksCount)
        {
            auto nodesBuilder = appender!(string[]);
            foreach(i; 0..nodesCount)
            {
                nodesBuilder.put(genUniqName());
            }

            auto nodes = nodesBuilder.data;
            auto edgeBuilder = appender!(IDirectedGraph.Edge[]);
            foreach(i; 0..linksCount)
            {
                size_t a = uniform(0, nodesCount);
                size_t b = uniform(0, nodesCount);
                edgeBuilder.put(IDirectedGraph.Edge(nodes[a], nodes[b], ""));
            }

            auto graph = new ConnListGraph;
            graph.load(edgeBuilder.data[].inputRangeObject);
            return graph;
        }

        static IDirectedGraph permuteGraph(IDirectedGraph graph, size_t permuteCount)
        {
            auto nodes = graph.nodes.array;
            IDirectedGraph.Edge[] edges;

            foreach(i; 0..permuteCount)
            {
                auto builder = appender!(IDirectedGraph.Edge[]);
                auto a = nodes[uniform(0, nodes.length)];
                auto b = nodes[uniform(0, nodes.length)];

                foreach(edge; graph.edges)
                {
                    if(a != b)
                    {
                        if(edge.source == a) edge.source = b;
                        else if(edge.source == b) edge.source = a;

                        if(edge.dist == a) edge.dist = b;
                        else if(edge.dist == b) edge.dist = a;
                    }

                    builder.put(edge);
                }

                edges = builder.data;
            }

            auto newGraph = new ConnListGraph;
            newGraph.load(edges.inputRangeObject);
            return newGraph;
        }
    }
}
```

15 Пакет EVOL.TYPES.ARGEDGE

```
module evol.types.argedge;

import std.conv;
import std.random;
import devol.serializable;
import devol.typemng;

import dyaml.all;

import graph.directed;

class ArgEdge : Argument, ISerializable
{
    this()
    {
        super( TypeMng.getSingleton().getType("TypeEdge") );
    }

    this(IDirectedGraph.IndexedEdge edge)
    {
        this();
        opAssign(edge);
    }

    ref ArgEdge opAssign(Argument val)
    {
        auto arg = cast(ArgEdge)(val);
        if (arg is null) return this;

        mEdge = arg.mEdge;
        return this;
    }

    ref ArgEdge opAssign(IDirectedGraph.IndexedEdge val)
    {
        mEdge = val;
        return this;
    }

    override @property string tostring(uint depth=0)
    {
        return to!string(mEdge);
    }

    @property IDirectedGraph.IndexedEdge edge()
    {
        return mEdge;
    }

    @property IDirectedGraph.IndexedEdge val()
    {
        return mEdge;
    }

    override void randomChange()
    {
        size_t permuteIndex(size_t i)
        {
            int change = uniform!"[]"(-2,2);
            if(cast(int)i + change < 0) return 0;
            return i + change;
        }

        auto chance = uniform!"[]"(0,1);
        if(chance == 0)
        {
            mEdge.source = permuteIndex(mEdge.source);
        } else if(chance == 1)
        {
            mEdge.dist = permuteIndex(mEdge.dist);
        }
    }

    override void randomChange(string maxChange)
    {
        randomChange();
    }

    override @property Argument dup()
    {
        auto darg = new ArgEdge();
        darg.mEdge = mEdge;
        return darg;
    }

    void saveBinary(OutputStream stream)
    {
        assert(false, "Not implemented!");
    }

    override Node saveYaml()
    {
        return Node([
            "class": Node("plain"),
            "source": Node(mEdge.source),
            "dist": Node(mEdge.dist),
            ]);
    }

    protected IDirectedGraph.IndexedEdge mEdge;
}
```

16 Пакет EVOL.TYPES.TYPEEDGE

```
module evol.types.typeedge;

import std.stream;

public
{
    import devol.argument;
    import devol.type;
    import evol.types.argedge;
}

import dyaml.all;

import graph.directed;

class TypeEdge : Type
{
    this()
    {
        super("TypeEdge");
    }

    override ArgEdge getNewArg()
    {
        auto arg = new ArgEdge();
        foreach(i; 0..10)
            arg.randomChange();

        return arg;
    }

    override ArgEdge getNewArg(string min, string max, string[] exVal)
    {
        return getNewArg;
    }

    override Argument loadArgument(InputStream stream)
    {
        assert(false, "Not implemented!");
    }

    override Argument loadArgument(Node node)
    {
        return new ArgEdge(IDirectedGraph.IndexedEdge(node["source"].as!size_t, node["dist"].as!size_t));
    }
}
```

17 Пакет EVOL.OPERATORS.AND

```
module evol.operators.and;

import devol.world;
import devol.std.line;
import devol.individ;
import devol.operator;
import devol.type;
import devol.typemng;
import devol.std.argpod;
import devol.std.typepod;
import devol.argument;

class AndOperator : Operator
{
    TypePod!bool booltype;

    this()
    {
        booltype = cast(TypePod!bool)TypeMng.getSingleton().getType("Typebool");
        mRetType = booltype;

        super("&&", Логическое" И'' длязначенийложьистина  /.", ArgsStyle.BINAR_STYLE);

        ArgInfo a1;
        a1.type = booltype;
        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract individ, Line line, WorldAbstract world)
    {
        auto ret = booltype.getNewArg();

        auto a1 = cast(ArgPod!bool)line[0];
        auto a2 = cast(ArgPod!bool)line[1];

        assert(a1 !is null);
        assert(a2 !is null);

        ret = a1.val && a2.val;
        return ret;
    }
}
```

18 Пакет EVOL.OPERATORS.ANSWER

```
module evol.operators.answer;

import devol.typemng;

import devol.individ;
import devol.world;
import devol.operator;
import devol.std.typepod;

import evol.individ;

class AnswerOperator : Operator
{
    TypePod!bool booltype;
    TypeVoid voidtype;

    enum description = Записывает" ответ. True − графыизоморфны , False − неизоморфны .";

    this()
    {
        booltype = cast(TypePod!bool)(TypeMng.getSingleton().getType("Typebool"));
        assert(booltype, "We need bool type!");

        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype);

        mRetType = voidtype;
        super("answer", description, ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = booltype;
        args ~= a1;
    }

    override Argument apply(IndAbstract aind, Line line, WorldAbstract world)
    {
        auto ind = cast(GraphIndivid)aind;
        assert(ind);

        auto cond = cast(ArgPod!bool)(line[0]);
        assert(cond);
        ind.answer = cond.val;

        return new ArgVoid;
    }
}
```

19 Пакет EVOL.OPERATORS.CONSTRUCT

```
module evol.operators.construct;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
    import devol.std.typepod;

    import evol.types.typeedge;
    import evol.types.argedge;
}

import evol.individ;
import graph.directed;

class ConstructOperator : Operator
{
    TypeEdge edgetype;
    TypePod!int inttype;

    enum description = Создает" новоеребрографаиздвухиндексов      : началаиконца  .";

    this()
    {
        edgetype = cast(TypeEdge)(TypeMng.getSingleton().getType("TypeEdge"));
        assert(edgetype, "We need edge type!");

        inttype = cast(TypePod!int)(TypeMng.getSingleton().getType("Typeint"));
        assert(inttype);

        mRetType = edgetype;
        super("construct", description, ArgsStyle.CLASSIC_STYLE);

        ArgInfo a1;
        a1.type = inttype;
        a1.max = "20";
        a1.min = "0";

        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        auto a1 = cast(ArgPod!int)(line[0]);
        auto a2 = cast(ArgPod!int)(line[0]);
        assert(a1);
        assert(a2);

        return new ArgEdge(IDirectedGraph.IndexedEdge(a1.val, a2.val));
    }
}
```

20 Пакет EVOL.OPERATORS.DIST

```
module evol.operators.dist;

import std.stdio;
import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
    import devol.std.typepod;

    import evol.types.typeedge;
    import evol.types.argedge;
}

import evol.individ;

class GetDistOperator : Operator
{
    TypeEdge edgetype;
    TypePod!int inttype;

    enum description = Возвращает" индексвершины , вкоторуюприходитребрографа    .";

    this()
    {
        edgetype = cast(TypeEdge)(TypeMng.getSingleton().getType("TypeEdge"));
        assert(edgetype, "We need edge type!");

        inttype = cast(TypePod!int)(TypeMng.getSingleton().getType("Typeint"));
        assert(inttype);

        mRetType = inttype;
        super("getDist", description, ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = edgetype;

        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        auto a1 = cast(ArgEdge)(line[0]);
        assert(a1);

        return new ArgPod!int(cast(int)a1.edge.dist);
    }
}
```

21 Пакет EVOL.OPERATORS.DIV

```
module evol.operators.div;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
}

class DivOperator : Operator
{
    TypePod!double doubletype;

    enum description = Арифметическая" операцияделениядействительныхчисел   .";

    this()
    {
        doubletype = cast(TypePod!double)(TypeMng.getSingleton().getType("Typedouble"));
        assert(doubletype, "We need double type!");

        mRetType = doubletype;
        super("/", description, ArgsStyle.BINAR_STYLE);
        ArgInfo a1;
        a1.type = doubletype;
        a1.min = "-1000";
        a1.max = "+1000";

        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto ret = doubletype.getNewArg();

        auto a1 = cast(ArgPod!double)(line[0]);
        auto a2 = cast(ArgPod!double)(line[1]);

        assert( a1 !is null, "Critical error: Operator plus, argument 1 isn't a right value!");
        assert( a2 !is null, "Critical error: Operator plus, argument 2 isn't a right value!");

        ret = a1.val / a2.val;
        return ret;
    }
}
```

22 Пакет EVOL.OPERATORS.GDUP

```
module evol.operators.gdup;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

class GenericDupOperator : Operator
{
    TypeVoid voidtype;

    enum description = Дублирует" головустекаобщегоназначения    .";

    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype, "We need double type!");

        mRetType = voidtype;
        super("gdup", description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        gind.genericStack.stackDup;
        return new ArgVoid;
    }
}
```

23 Пакет EVOL.OPERATORS.GOVER

```
module evol.operators.gover;

import std.stdio;

import devol.typemng;

public
```

```
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

class GenericOverOperator : Operator
{
    TypeVoid voidtype;

    enum description = Дублирует" значениеподголовойстеканавершину     . Стекобщегоназначения  .";

    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype, "We need double type!");

        mRetType = voidtype;
        super("gover", description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        gind.genericStack.stackOver;
        return new ArgVoid;
    }
}
```

24 Пакет EVOL.OPERATORS.GPOP

```
module evol.operators.gpop;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
}

import evol.individ;

class GenericPopOperator : Operator
{
    TypePod!double doubletype;

    enum description = Взятие" головысостекаобщегоназначения    .";

    this()
    {
        doubletype = cast(TypePod!double)(TypeMng.getSingleton().getType("Typedouble"));
        assert(doubletype, "We need double type!");

        mRetType = doubletype;
        super("gpop", description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        return gind.genericStack.stackPop;
    }
}
```

25 Пакет EVOL.OPERATORS.GPUSH

```
module evol.operators.gpush;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

class GenericPushOperator : Operator
{
    TypePod!double doubletype;
    TypeVoid voidtype;

    enum description = Сохраняет" действительноечисловстекобщегоназначения      .";

    this()
    {
        doubletype = cast(TypePod!double)(TypeMng.getSingleton().getType("Typedouble"));
        assert(doubletype, "We need double type!");

        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype);

        mRetType = voidtype;
        super("gpush", description, ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = doubletype;
        a1.min = "-1000";
        a1.max = "+1000";

        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        auto a1 = cast(ArgPod!double)(line[0]);
        assert(a1);

        gind.genericStack.stackPush(a1);

        return new ArgVoid;
    }
}
```

26 Пакет EVOL.OPERATORS.GROT

```
module evol.operators.grot;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

class GenericRotOperator : Operator
{
    TypeVoid voidtype;

    enum description = Перемещает" третийэлементсголовыстеканавершину       . Стекобщегоназначения  .";
    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype, "We need double type!");

        mRetType = voidtype;
        super("grot", description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        gind.genericStack.stackRot;
        return new ArgVoid;
    }
}
```

27 Пакет EVOL.OPERATORS.GSWAP

```
module evol.operators.gswap;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

class GenericSwapOperator : Operator
{
    TypeVoid voidtype;

    enum description = Меняет" местамидвапоследнихзначениявстекеобщегоназначения ." ;

    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype, "We need double type!");

        mRetType = voidtype;
        super("gswap", description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        gind.genericStack.stackSwap;
        return new ArgVoid;
    }
}
```

28 Пакет EVOL.OPERATORS.IDCAST

```
module evol.operators.idcast;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
}

class IntDoubleCastOperator : Operator
{
    TypePod!double doubletype;
    TypePod!int inttype;

    enum description = Преобразует" целочисленноевдействительноечисло    .";

    this()
    {
        inttype = cast(TypePod!int)(TypeMng.getSingleton().getType("Typeint"));
        assert(inttype, "We need int type!");

        doubletype = cast(TypePod!double)(TypeMng.getSingleton().getType("Typedouble"));
        assert(doubletype, "We need double type!");

        mRetType = doubletype;
        super("cast", description, ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = inttype;
        a1.min = "-100";
        a1.max = "+100";

        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto ret = doubletype.getNewArg();

        auto a1 = cast(ArgPod!int)(line[0]);

        assert( a1 !is null, "Critical error: Operator plus, argument 1 isn't a right value!");

        ret = cast(double)a1.val;
        return ret;
    }
}
```

29 Пакет EVOL.OPERATORS.IDUP

```
module evol.operators.idup;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

alias InputDupFirstOperator = InputDupOperator!((ind) => ind.firstGraphStack, "idup1"
    , Копирует" вершинустекапервогографа    .");
alias InputDupSecondOperator = InputDupOperator!((ind) => ind.secondGraphStack, "idup2"
    , Копирует" вершинустекавторогографа    .");

class InputDupOperator(alias stack, string opname, string description) : Operator
{
    TypeVoid voidtype;

    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype);

        mRetType = voidtype;
        super(opname, description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);
        stack(gind).stackDup();

        return new ArgVoid;
    }
}
```

30 Пакет EVOL.OPERATORS.IOVER

```
module evol.operators.iover;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

alias InputOverFirstOperator = InputOverOperator!((ind) => ind.firstGraphStack, "iover1"
    , Копирует" второйэлементстекапервогографанавершину     .");
alias InputOverSecondOperator = InputOverOperator!((ind) => ind.secondGraphStack, "iover2"
    , Копирует" второйэлементстекавторогографанавершину     .");

class InputOverOperator(alias stack, string opname, string description) : Operator
{
    TypeVoid voidtype;

    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype);

        mRetType = voidtype;
        super(opname, description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        stack(gind).stackOver();

        return new ArgVoid;
    }
}
```

31 Пакет EVOL.OPERATORS.IPOP

```
module evol.operators.ipop;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import evol.types.typeedge;
    import evol.types.argedge;
}

import evol.individ;

alias InputPopFirstOperator = InputPopOperator!((ind) => ind.firstGraphStack, "ipop1",Снимает
    " ивозвращаетголовустекапервогографа     .");
alias InputPopSecondOperator = InputPopOperator!((ind) => ind.secondGraphStack, "ipop2",Снимает
    " ивозвращаетголовустекавторогографа     .");
```

```
class InputPopOperator(alias stack, string opname, string description) : Operator
{
    TypeEdge edgetype;

    this()
    {
        edgetype = cast(TypeEdge)(TypeMng.getSingleton().getType("TypeEdge"));
        assert(edgetype, "We need edge type!");

        mRetType = edgetype;
        super(opname, description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        return stack(gind).stackPop;
    }
}
```

32 Пакет EVOL.OPERATORS.IPUSH

```
module evol.operators.ipush;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;

    import evol.types.typeedge;
    import evol.types.argedge;
}

import evol.individ;

alias InputPushFirstOperator = InputPushOperator!((ind) => ind.firstGraphStack, "ipush1"
    , Сохраняет" граньграфавовходнойстекдляпервогографа        ");
alias InputPushSecondOperator = InputPushOperator!((ind) => ind.secondGraphStack, "ipush2"
    , Сохраняет" граньграфавовходнойстекдлявторогографа        ");

class InputPushOperator(alias stack, string opname, string description) : Operator
{
    TypeEdge edgetype;
    TypeVoid voidtype;

    this()
    {
        edgetype = cast(TypeEdge)(TypeMng.getSingleton().getType("TypeEdge"));
        assert(edgetype, "We need edge type!");

        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype);

        mRetType = voidtype;
        super(opname, description, ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = edgetype;

        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        auto a1 = cast(ArgEdge)(line[0]);
        assert(a1);

        stack(gind).stackPush(a1);
        return new ArgVoid;
    }
}
```

33 Пакет EVOL.OPERATORS.IROT

```
module evol.operators.irot;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

alias InputRotFirstOperator = InputRotOperator!((ind) => ind.firstGraphStack, "irot1"
    , Перемещает" третийэлементстекапервогографанавершину      .");
alias InputRotSecondOperator = InputRotOperator!((ind) => ind.secondGraphStack, "irot2"
    , Перемещает" третийэлементстекавторогографанавершину      .");

class InputRotOperator(alias stack, string opname, string description) : Operator
{
    TypeVoid voidtype;

    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype);

        mRetType = voidtype;
        super(opname, description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        stack(gind).stackRot();

        return new ArgVoid;
    }
}
```

34 Пакет EVOL.OPERATORS.ISWAP

```
module evol.operators.iswap;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
}

import evol.individ;

alias InputSwapFirstOperator = InputSwapOperator!((ind) => ind.firstGraphStack, "iswap1"
    , Меняет" первыхдваэлементанастекедляпервогографаместами        .");
alias InputSwapSecondOperator = InputSwapOperator!((ind) => ind.secondGraphStack, "iswap2"
    , Меняет" первыхдваэлементанастекедлявторогографаместами        .");

class InputSwapOperator(alias stack, string opname, string description) : Operator
{
    TypeVoid voidtype;

    this()
    {
        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));
        assert(voidtype);

        mRetType = voidtype;
        super(opname, description, ArgsStyle.NULAR_STYLE);
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        stack(gind).stackSwap();

        return new ArgVoid;
    }
}
```

35 Пакет EVOL.OPERATORS.MULT

```
module evol.operators.mult;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
}

class MultOperator : Operator
{
    TypePod!double doubletype;

    enum description = Арифметическая" операцияумножениядействительныхчисел    .";

    this()
    {
        doubletype = cast(TypePod!double)(TypeMng.getSingleton().getType("Typedouble"));
        assert(doubletype, "We need double type!");

        mRetType = doubletype;
        super("*", description, ArgsStyle.BINAR_STYLE);

        ArgInfo a1;
        a1.type = doubletype;
        a1.min = "-1000";
        a1.max = "+1000";

        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto ret = doubletype.getNewArg();

        auto a1 = cast(ArgPod!double)(line[0]);
        auto a2 = cast(ArgPod!double)(line[1]);

        assert( a1 !is null, "Critical error: Operator plus, argument 1 isn't a right value!");
        assert( a2 !is null, "Critical error: Operator plus, argument 2 isn't a right value!");

        ret = a1.val * a2.val;
        return ret;
    }
}
```

36 Пакет EVOL.OPERATORS.NOT

```
module evol.operators.not;

import devol.world;
import devol.std.line;
import devol.individ;
import devol.operator;
import devol.type;
import devol.typemng;
import devol.std.argpod;
import devol.std.typepod;
import devol.argument;

class NotOperator : Operator
{
    TypePod!bool booltype;

    this()
    {
        booltype = cast(TypePod!bool)TypeMng.getSingleton().getType("Typebool");
        mRetType = booltype;

        super("!", Логическое" НЕТ'' длязначенийложьистина  /.", ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = booltype;
        args ~= a1;
    }

    override Argument apply(IndAbstract individ, Line line, WorldAbstract world)
    {
        auto ret = booltype.getNewArg();

        auto a1 = cast(ArgPod!bool)line[0];

        assert(a1 !is null);

        ret = !a1.val;
        return ret;
    }
}
```

37 Пакет EVOL.OPERATORS.OPIF

```
module evol.operators.opif;

import devol.typemng;

import devol.individ;
import devol.world;
import devol.operator;
import devol.std.typepod;

debug import std.stdio;

class IfOperator : Operator
{
    TypePod!bool booltype;
    TypeVoid voidtype;

    enum description = Условный" оператор, которыйберettриаргумента  . Первый "имеет
    " логическийтип , которыйотноситсякусловиюдействия  . Еслиэтотаргумент  "вычисляется
    " взначениеИСТИНА  '', товозвращаетсявторойаргумент  , иначе "возвращается
    " третийаргумент . Второйитретийаргументыотносятсякдействиям  , "которые
    " имеюттип  void";

    this()
    {
        booltype = cast(TypePod!bool)(TypeMng.getSingleton().getType("Typebool"));
        assert(booltype, "We need bool type!");

        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));

        mRetType = voidtype;
        super("if", description, ArgsStyle.CONTROL_STYLE);

        ArgInfo a1;
        a1.type = booltype;
        args ~= a1;

        a1.type = voidtype;
        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto cond = cast(ArgPod!bool)(line[0]);

        Line vthen = cast(Line)(line[1]);
        Line velse = cast(Line)(line[2]);

        ArgScope sthen = cast(ArgScope)(line[1]);
        ArgScope selse = cast(ArgScope)(line[2]);

        if (cond.val)
        {
            if (vthen !is null)
            {
                vthen.compile(ind, world);
            } else if (sthen !is null)
            {
                foreach(Line aline; sthen)
                {
                    auto line = cast(Line)aline;
                    line.compile(ind, world);
                }
            } else
            {
                debug writeln("Warning: invalid ThenArg: ", line.tostring);
            }//else throw new Exception("If is confused! ThenArg is no line, no scope. " ~ line.tostring);
        } else
        {
            if (velse !is null)
            {
                velse.compile(ind, world);
            } else if (selse !is null)
            {
                foreach(Line aline; selse)
                {
                    auto line = cast(Line)aline;
                    line.compile(ind, world);
                }
            } else
            {
                debug writeln("Warning: invalid ElseArg: ", line.tostring);
            } //else throw new Exception("If is confused! ElseArg is no line, no scope" ~ line.tostring);
        }
        return voidtype.getNewArg();
    }
}
```

38 Пакет EVOL.OPERATORS.OPWHILE

```
module evol.operators.opwhile;

import devol.typemng;

import devol.individ;
import devol.world;
import devol.operator;
import devol.std.typepod;

debug import std.stdio;

class WhileOperator : Operator
{
    TypePod!bool booltype;
    TypeVoid voidtype;

    enum MAX_ITERATIONS = 100;

    enum description = Оператор", управляющийпотокомисполнения  . Еговторойаргумент   "выполняется
                       " дотехпор  , покапервыйаргументвычисляетсявИСТИНА     . "Во
                       " избежаниебесконечнойпрограммынакладываетсяограничениена      "максимальное
                       " числоитераций .";

    this()
    {
        booltype = cast(TypePod!bool)(TypeMng.getSingleton().getType("Typebool"));
        assert(booltype, "We need bool type!");

        voidtype = cast(TypeVoid)(TypeMng.getSingleton().getType("TypeVoid"));

        mRetType = voidtype;
        super("while", description, ArgsStyle.CONTROL_STYLE);

        ArgInfo a1;
        a1.type = booltype;
        a1.eval = false;
        args ~= a1;

        a1.type = voidtype;
        a1.eval = false;
        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto condLine = cast(Line)(line[0]);
        auto condConst = cast(ArgPod!bool)(line[0]);
        size_t iterations;

        Line vaction = cast(Line)(line[1]);
        ArgScope saction = cast(ArgScope)(line[1]);

        void iterateOnce()
        {
            if (vaction !is null)
            {
                vaction.compile(ind, world);
            } else if (saction !is null)
            {
                foreach(Line aline; saction)
                {
                    auto line = cast(Line)aline;
                    line.compile(ind, world);
                }
            } else
            {
                debug writeln("Warning: invalid ThenArg: ", line.tostring);
            }
        }

        if(condLine is null)
        {
            assert(condConst);
            if(condConst)
            {
               foreach(i; 0..MAX_ITERATIONS)
               {
                   iterateOnce();
               }
            }
        } else
        {
           foreach(i; 0..MAX_ITERATIONS)
           {
               if(!condLine.compile(ind, world)) break;
               iterateOnce();
           }
        }

        return voidtype.getNewArg();
    }
}
```

39 Пакет EVOL.OPERATORS.OR

```
module evol.operators.or;

import devol.world;
import devol.std.line;
import devol.individ;
import devol.operator;
import devol.type;
import devol.typemng;
import devol.std.argpod;
import devol.std.typepod;
import devol.argument;

class OrOperator : Operator
{
    TypePod!bool booltype;

    this()
    {
        booltype = cast(TypePod!bool)TypeMng.getSingleton().getType("Typebool");
        mRetType = booltype;

        super("||", Логическое" ИЛИ'' длязначенийложьистина  /.", ArgsStyle.BINAR_STYLE);

        ArgInfo a1;
        a1.type = booltype;
        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract individ, Line line, WorldAbstract world)
    {
        auto ret = booltype.getNewArg();

        auto a1 = cast(ArgPod!bool)line[0];
        auto a2 = cast(ArgPod!bool)line[1];

        assert(a1 !is null);
        assert(a2 !is null);

        ret = a1.val || a2.val;
        return ret;
    }
}
```

40 Пакет EVOL.OPERATORS.PLUS

```
module evol.operators.plus;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
}

class PlusOperator : Operator
{
    TypePod!double doubletype;

    enum description = Арифметическая" операциясложениядействительныхчисел   .";

    this()
    {
        doubletype = cast(TypePod!double)(TypeMng.getSingleton().getType("Typedouble"));
        assert(doubletype, "We need double type!");

        mRetType = doubletype;
        super("+", description, ArgsStyle.BINAR_STYLE);

        ArgInfo a1;
        a1.type = doubletype;
        a1.min = "-1000";
        a1.max = "+1000";

        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto ret = doubletype.getNewArg();

        auto a1 = cast(ArgPod!double)(line[0]);
        auto a2 = cast(ArgPod!double)(line[1]);

        assert( a1 !is null, "Critical error: Operator plus, argument 1 isn't a right value!");
        assert( a2 !is null, "Critical error: Operator plus, argument 2 isn't a right value!");

        ret = a1.val + a2.val;
        return ret;
    }
}
```

41 Пакет EVOL.OPERATORS.RELATION

```
module evol.operators.relation;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
}

alias IntEqualOperator = RelationOperator!("== (int)", "==", TypePod!int, ArgPod!int, "Typeint", Сравнение"
    ↪ наравенствоцелочисленныхаргументов   .");
alias DoubleEqualOperator = RelationOperator!("== (double)", "==", TypePod!double, ArgPod!double, "Typedouble",
    ↪ Сравнение" наравенстводействительныхаргументов   .");

alias IntGreaterOperator = RelationOperator!("> (int)", ">", TypePod!int, ArgPod!int, "Typeint", Сравнение"
    ↪ целочисленныхаргументов . ВозвращаетИСТИНА , еслипервыйбольшевторого   .");
alias IntLesserOperator = RelationOperator!("< (int)", "<", TypePod!int, ArgPod!int, "Typeint", Сравнение"
    ↪ целочисленныхаргументов . ВозвращаетИСТИНА , еслипервыйменьшевторого   .");
alias IntGreaterEqualOperator = RelationOperator!(">= (int)", ">=", TypePod!int, ArgPod!int, "Typeint", Сравнение"
    ↪ целочисленныхаргументов . ВозвращаетИСТИНА , еслипервыйбольшевторогоилиравенвторому      .");
alias IntLesserEqualOperator = RelationOperator!("<= (int)", "<=", TypePod!int, ArgPod!int, "Typeint", Сравнение"
    ↪ целочисленныхаргументов . ВозвращаетИСТИНА , еслипервыйменьшевторогоилиравенвторому      .");

alias DoubleGreaterOperator = RelationOperator!("> (double)", ">", TypePod!double, ArgPod!double, "Typedouble",
    ↪ Сравнение" действительныхаргументов . ВозвращаетИСТИНА , еслипервыйбольшевторого   .");
alias DoubleLesserOperator = RelationOperator!("< (double)", "<", TypePod!double, ArgPod!double, "Typedouble",
    ↪ Сравнение" действительныхаргументов . ВозвращаетИСТИНА , еслипервыйменьшевторого   .");

class RelationOperator(string opname, string relation, DslType, DslArg, string dslTypeName, string description) :
    ↪ Operator
{
    DslType inputType;
    TypePod!bool boolType;

    static assert(opname != "");

    this()
    {
        inputType = cast(DslType)(TypeMng.getSingleton().getType(dslTypeName));
        assert(inputType, "We need "~dslTypeName~" type!");

        boolType = cast(TypePod!bool)(TypeMng.getSingleton().getType("Typebool"));
        assert(boolType, "We need bool type!");

        mRetType = boolType;
        super(opname, description, ArgsStyle.BINAR_STYLE);

        ArgInfo a1;
        a1.type = inputType;
        a1.min = "-1000";
        a1.max = "+1000";

        args ~= a1;
        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto ret = boolType.getNewArg();

        auto a1 = cast(DslArg)(line[0]);
        auto a2 = cast(DslArg)(line[1]);

        assert( a1 !is null, "Critical error: Operator "~name~", argument 1 isn't a right value!");
        assert( a2 !is null, "Critical error: Operator "~name~", argument 2 isn't a right value!");

        ret = mixin(q{a1.val } ~ relation ~ q{ a2.val});
        return ret;
    }
}
```

42 Пакет EVOL.OPERATORS.ROUND

```
module evol.operators.round;

import std.stdio;
import std.math;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typepod;
}

class RoundOperator : Operator
{
    TypePod!double doubletype;
    TypePod!int inttype;

    enum description = Преобразует" действительноевцелочисленноечислоспомощьюматематическогоокругления        .";

    this()
    {
        inttype = cast(TypePod!int)(TypeMng.getSingleton().getType("Typeint"));
        assert(inttype, "We need int type!");

        doubletype = cast(TypePod!double)(TypeMng.getSingleton().getType("Typedouble"));
        assert(doubletype, "We need double type!");

        mRetType = inttype;
        super("round", description, ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = doubletype;
        a1.min = "-100";
        a1.max = "+100";

        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto ret = inttype.getNewArg();

        auto a1 = cast(ArgPod!double)(line[0]);

        assert( a1 !is null, "Critical error: Operator plus, argument 1 isn't a right value!");

        ret = cast(int)round(a1.val);
        return ret;
    }
}
```

43 Пакет EVOL.OPERATORS.SOURCE

```
module evol.operators.source;

import std.stdio;

import devol.typemng;

public
{
    import devol.individ;
    import devol.world;
    import devol.operator;
    import devol.std.typevoid;
    import devol.std.argvoid;
    import devol.std.typepod;

    import evol.types.typeedge;
    import evol.types.argedge;
}

import evol.individ;

class GetSourceOperator : Operator
{
    TypeEdge edgetype;
    TypePod!int inttype;
    enum description = Возвращает" индексвершины , изкоторойвыходитребрографа ."; 

    this()
    {
        edgetype = cast(TypeEdge)(TypeMng.getSingleton().getType("TypeEdge"));
        assert(edgetype, "We need edge type!");

        inttype = cast(TypePod!int)(TypeMng.getSingleton().getType("Typeint"));
        assert(inttype);

        mRetType = inttype;
        super("getSource", description, ArgsStyle.UNAR_STYLE);

        ArgInfo a1;
        a1.type = edgetype;

        args ~= a1;
    }

    override Argument apply(IndAbstract ind, Line line, WorldAbstract world)
    {
        auto gind = cast(GraphIndivid)ind;
        assert(gind);

        auto a1 = cast(ArgEdge)(line[0]);
        assert(a1);

        return new ArgPod!int(cast(int)a1.edge.source);
    }
}
```