# Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

# Федеральное государственное образовательное бюджетное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (ФГБОУ «СибГУТИ»)

Кафедра ПМ и К

**Курсовая работа**

Дисциплина: Визуальное программирование и человеко-машинное взаимодействие

Тема: «Приложение-симулятор логических схем»

Вариант 21

Выполнил:

Студент группы ИП-117

Шириков А.С.

Проверил:

Ассистент кафедры ПМ и К

Меркулов И. А.

Новосибирск 2023

**Содержание**

[Задание 3](file:///C:\Users\shiri\Downloads\Otchet_po_kursovoy_rabote_TRPO_Shirikov_A_S.docx#__RefHeading___Toc132_1766323042)

[Пример работы программы 4](file:///C:\Users\shiri\Downloads\Otchet_po_kursovoy_rabote_TRPO_Shirikov_A_S.docx#__RefHeading___Toc134_1766323042)

[Unit-тесты 5](file:///C:\Users\shiri\Downloads\Otchet_po_kursovoy_rabote_TRPO_Shirikov_A_S.docx#__RefHeading___Toc136_1766323042)

[Use-Case диаграмма 5](file:///C:\Users\shiri\Downloads\Otchet_po_kursovoy_rabote_TRPO_Shirikov_A_S.docx#__RefHeading___Toc136_1766323042)

Заключение 6

[Листинг программы](file:///C:\Users\shiri\Downloads\Otchet_po_kursovoy_rabote_TRPO_Shirikov_A_S.docx#__RefHeading___Toc138_1766323042) 7

**Задание**

Главное окно приложения состоит из холста, на котором размещаются логические элементы; дерева проекта, на котором отображается список схем проекта; панели элементов на котором располагаются логические элементы, которые можно добавлять на схему; меню управления из которого можно сохранить текущий проект, загрузить существующий проект, выйти из программы. Для размещения элемента на схеме, нужно выбрать логический элемент на панели и кликнуть левой кнопкой мыши на холст в то место на которое нужно поместить логический элемент. При зажатой левой кнопкой мыши на логическом элементе, расположенном на холсте, его можно перетаскивать. Для соединения логических элементов необходимо перетащить выход одного элемента на вход другого элемента или наоборот. При этом после соединения появляется линия между входом и выходом соединённых элементов. Для удаления соединения необходимо выбрать линию соединения левой кнопкой мыши и нажать клавишу delete. Для удаления элемента, необходимо выбрать удаляемый элемент левой кнопкой мыши и нажать клавишу delete. Выходные сигналы логических элементов должны рассчитываться в реальном времени. Проект имеет вид списка с верхним элементом - названием проекта, все остальные элементы названия схем. Схемы можно добавлять и удалять, но в проекте всегда должна быть минимум одна схема. Чтобы отредактировать схему нужно кликнуть на неё в списке два раза левой кнопкой мыши. Название проекта можно отредактировать кликнув два раза левой кнопкой мыши на нём. Меню должно включать четыре пункта: "Создать", "Открыть", "Сохранить", "выйти". Пункт "Создать" - создаёт новый проект. Пункт "Открыть" - открывает диалоговое окно открытия файла проекта, при выборе файла проекта, проект подгружается в приложение - его схемы отображаются в списке схем проекта, на холсте появляется отображение первой схемы в проекте. Пункт "Сохранить" - открывает диалоговое окно сохранения проекта в файл. При выборе файла содержимое проекта сохраняется в него. Пункт "Выйти" - закрывает приложение. При старте приложения появляется окно в котором отображается список недавно открывавшихся проектов, отсортированных по дате открытия. Также есть кнопки "Создать новый проект", "Открыть проект", "Выйти", которые дублируют пункты меню "Создать", "Открыть", "Выйти" основного окна. При выборе проекта из списка стартового окна, он открывается в главном окне. Поддерживаемые логические элементы Приложение должно иметь поддержку логических элементов И, ИЛИ, НЕ, ИСКЛ-ИЛИ, а также элементов ВХОД и ВЫХОД. ВХОД позволяет по клику на нему левой кнопкой мыши поменять выходящее из него значение, ВЫХОД отображает значение сигнала, подающееся ему на вход.

**Пример работы программы**

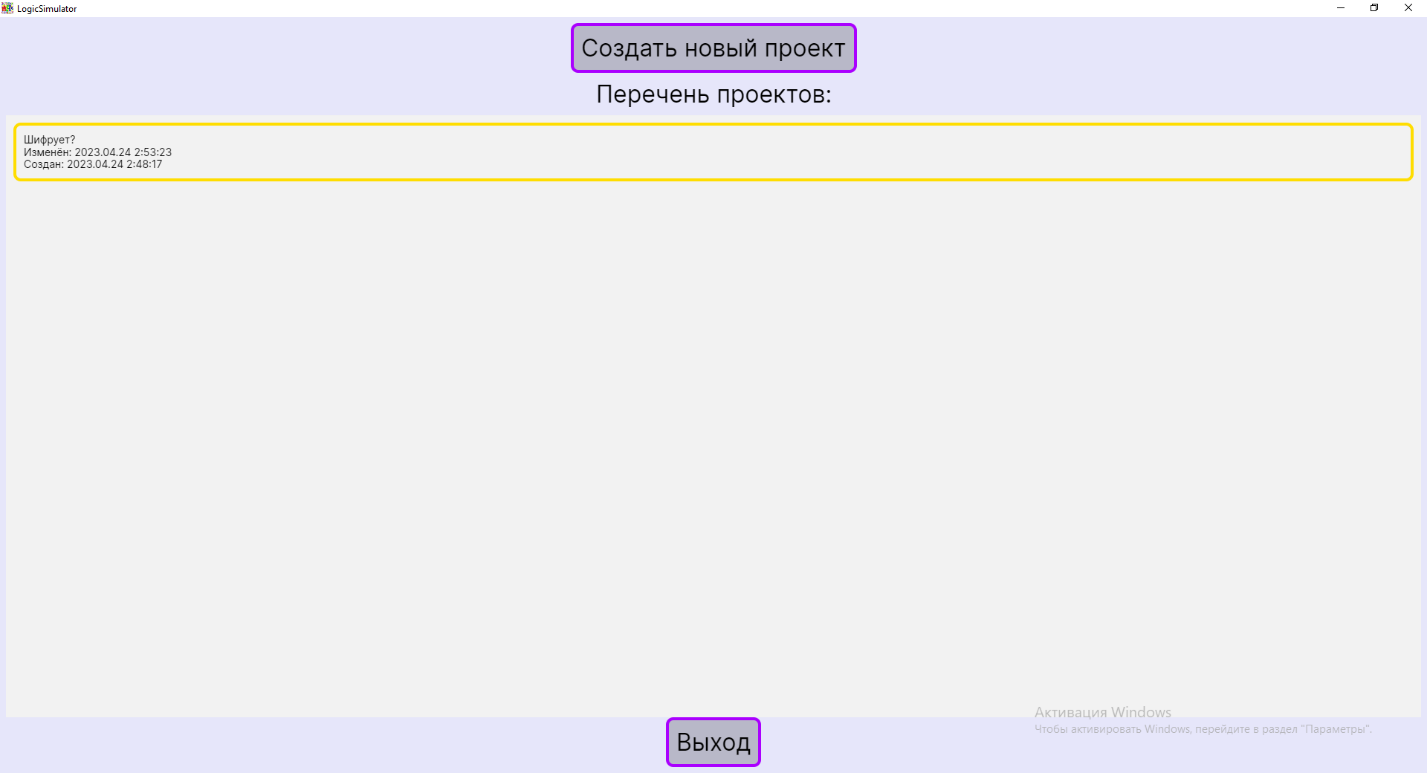


Рисунок 1 Окно открытия проекта

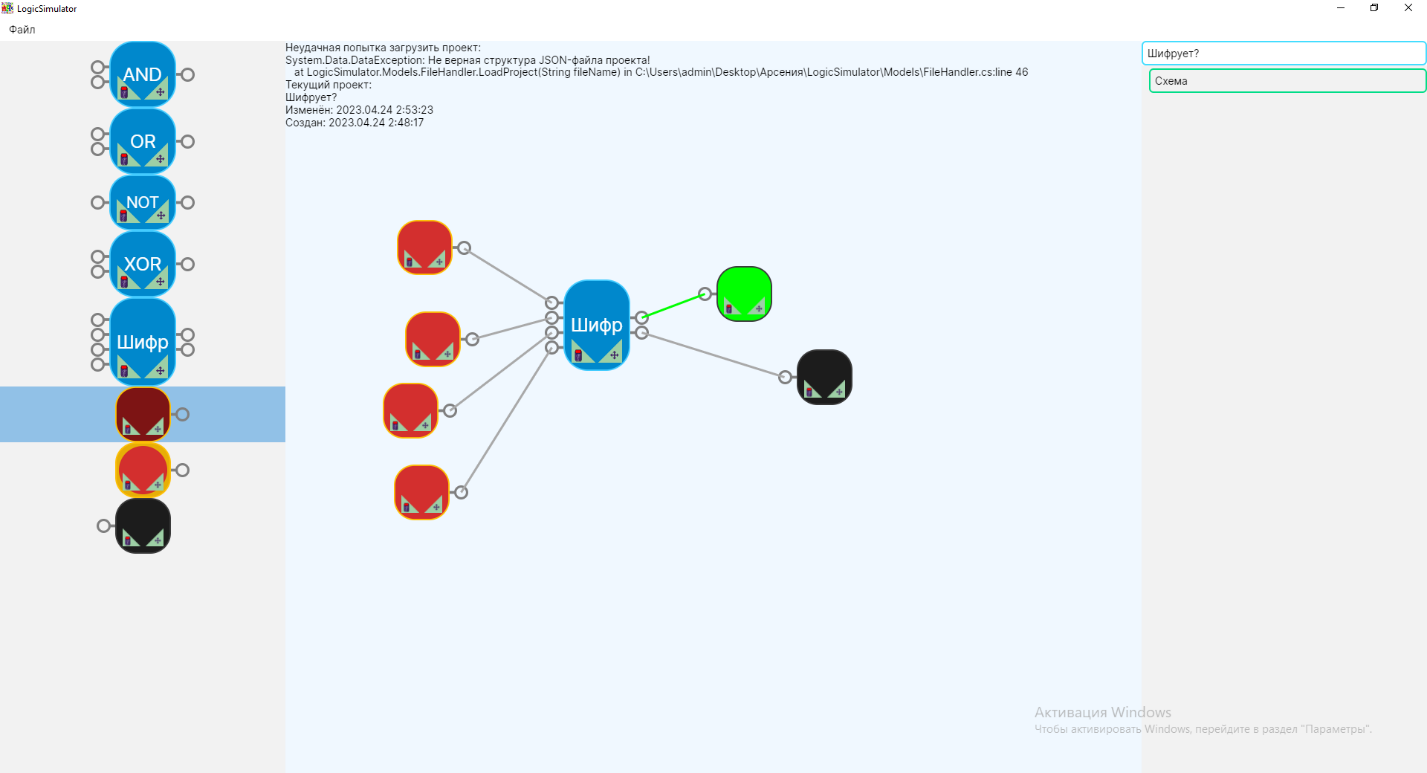


Рисунок 2 Процесс работы со схемой

**Unit-тесты**

В процессе модульного тестирования проверялась работоспособность ключевых разделов программы, таких как окно создания/выбора проекта, а также непосредственно среда создания электронной схемы. Проверка осуществляется на предмет соответствия кода подученной схемы ожидаемым параметрам.

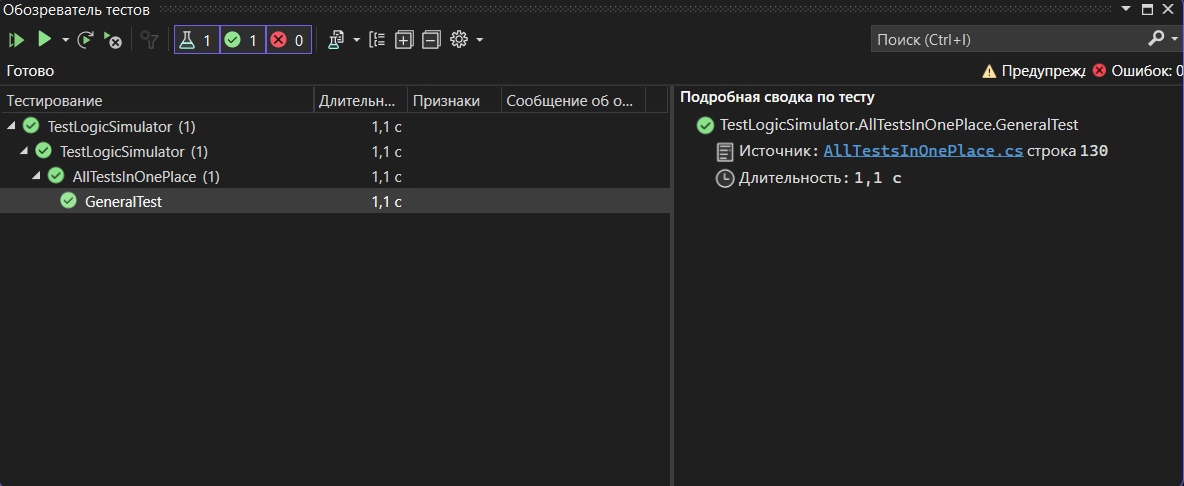
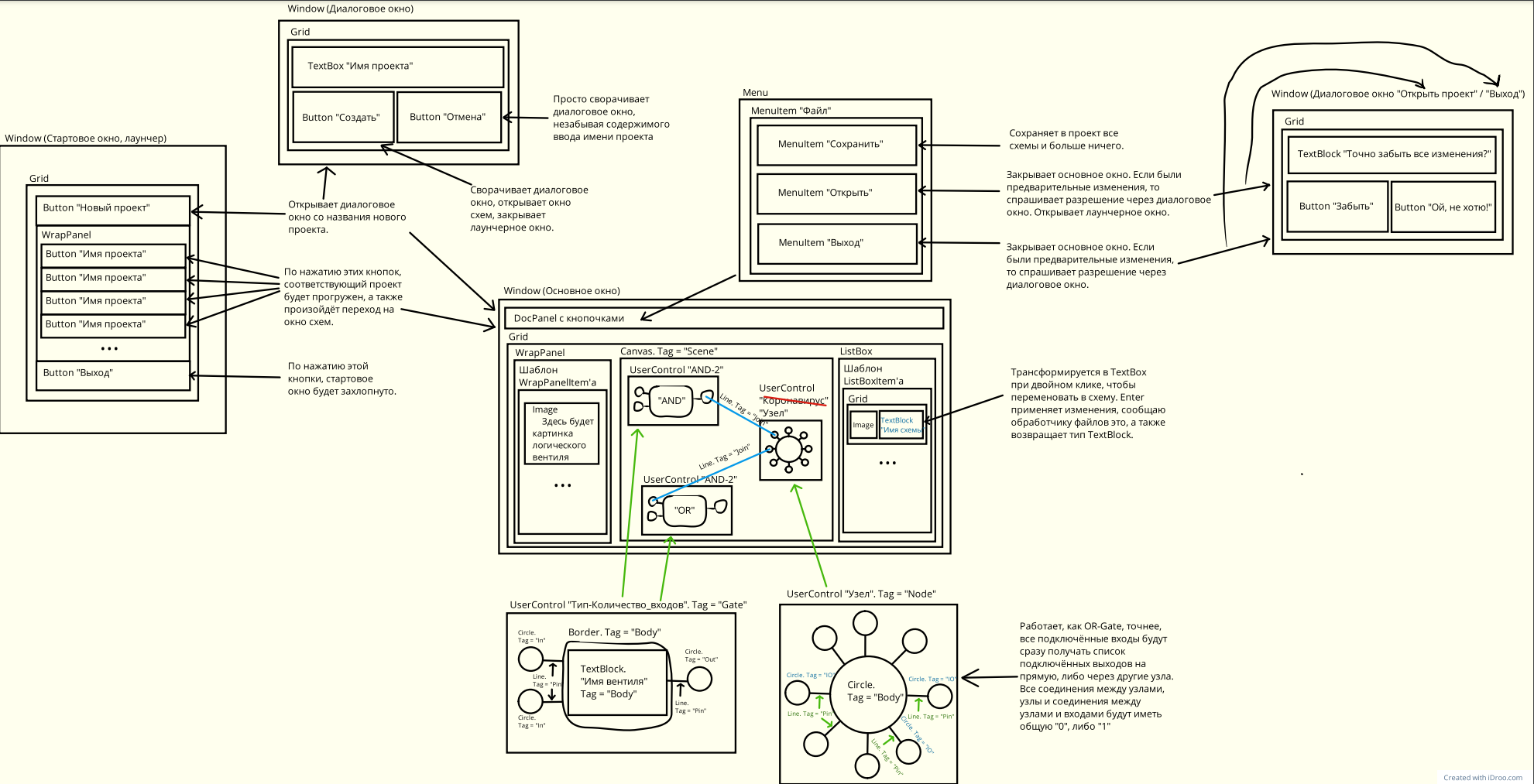


Рисунок 3 Результат выполнения тестов

**Use-case диаграмма**



**Заключение**

Разработанный проект представляет собой полноценное приложение для работы с логическими схемами, в котором имеются несколько видов элементов и возможность объединять их между собой. Кроме того, предусмотрена возможность сохранения проектов и работы в различных форматах. При создании проекта применены современные технологии разработки программ с графическим интерфейсом.

**Листинг программы**

**Models/destinator.cs**

using Avalonia;

using LogicSimulator.Views.Shapes;

namespace LogicSimulator.Models {

public class Distantor {

public readonly int num;

public IGate parent;

public readonly string tag;

readonly Visual? ref\_point;

public Distantor(IGate parent, int n, Visual? r\_p, string tag) {

this.parent = parent;

num = n; // Например, в AND\_2-gate'е: 0 и 1 - входы, 2 - выход

ref\_point = r\_p;

this.tag = tag;

}

public Point GetPos() => parent.GetPinPos(num, ref\_point);

}

}

**Models/mapper.cs**

using Avalonia.Controls;

using Avalonia;

using LogicSimulator.ViewModels;

using LogicSimulator.Views.Shapes;

using System;

using System.Collections.Generic;

using DynamicData;

using Avalonia.Controls.Shapes;

using Avalonia.Media;

using Avalonia.LogicalTree;

using System.Linq;

using Button = LogicSimulator.Views.Shapes.Button;

using System.Threading.Tasks;

using Avalonia.Threading;

namespace LogicSimulator.Models {

public class Mapper {

readonly Line marker = new() { Tag = "Marker", ZIndex = 2, IsVisible = false, Stroke = Brushes.YellowGreen, StrokeThickness = 3 };

public Line Marker { get => marker; }

readonly Simulator sim = new();

/\*

\* Выборка элементов

\*/

private int selected\_item = 0;

public int SelectedItem { get => selected\_item; set => selected\_item = value; }

private static IGate CreateItem(int n) {

return n switch {

0 => new AND\_2(),

1 => new OR\_2(),

2 => new NOT(),

3 => new XOR\_2(),

4 => new Shifrator(),

5 => new Switch(),

6 => new Button(),

7 => new LightBulb(),

\_ => new AND\_2(),

};

}

public IGate[] item\_types = new IGate[] {

CreateItem(0),

CreateItem(1),

CreateItem(2),

CreateItem(3),

CreateItem(4),

CreateItem(5),

CreateItem(6),

CreateItem(7),

};

public IGate GenSelectedItem() => CreateItem(selected\_item);

/\*

\* Хранилище

\*/

readonly List<IGate> items = new();

public void AddItem(IGate item) {

items.Add(item);

sim.AddItem(item);

}

public void RemoveItem(IGate item) {

items.Remove(item);

sim.RemoveItem(item);

item.ClearJoins();

((Control) item).Remove();

}

public void RemoveAll() {

foreach (var item in items.ToArray()) RemoveItem(item);

}

/\*

\* Определение режима перемещения

\*/

int mode = 0;

/\*

\* Режимы:

\* 0 - ничего не делает

\* 1 - двигаем камеру

\* 2 - двигаем элемент

\* 3 - тянем элемент

\* 4 - вышвыриваем элемент

\* 5 - тянем линию от входа (In)

\* 6 - тянем линию от выхода (Out)

\* 7 - тянем линию от узла (IO)

\* 8 - тянем уже существующее соединение - переподключаем

\*/

private static int CalcMode(string? tag) {

if (tag == null) return 0;

return tag switch {

"Scene" => 1,

"Body" => 2,

"Resizer" => 3,

"Deleter" => 4,

"In" => 5,

"Out" => 6,

"IO" => 7,

"Join" => 8,

"Pin" or \_ => 0,

};

}

private void UpdateMode(Control item) => mode = CalcMode((string?) item.Tag);

private static bool IsMode(Control item, string[] mods) {

var name = (string?) item.Tag;

if (name == null) return false;

return mods.IndexOf(name) != -1;

}

private static UserControl? GetUC(Control item) {

while (item.Parent != null) {

if (item is UserControl @UC) return @UC;

item = (Control) item.Parent;

}

return null;

}

private static IGate? GetGate(Control item) {

var UC = GetUC(item);

if (UC is IGate @gate) return @gate;

return null;

}

/\*

\* Обработка мыши

\*/

Point moved\_pos;

IGate? moved\_item;

Point item\_old\_pos;

Size item\_old\_size;

Ellipse? marker\_circle;

Distantor? start\_dist;

int marker\_mode;

Line? old\_join;

bool join\_start;

bool delete\_join = false;

public void Press(Control item, Point pos) {

// Log.Write("PointerPressed: " + item.GetType().Name + " pos: " + pos);

UpdateMode(item);

// Log.Write("new\_mode: " + mode);

moved\_pos = pos;

moved\_item = GetGate(item);

tapped = true;

if (moved\_item != null) item\_old\_pos = moved\_item.GetPos();

switch (mode) {

case 3:

if (moved\_item == null) break;

item\_old\_size = moved\_item.GetBodySize();

break;

case 5 or 6 or 7:

if (marker\_circle == null) break;

var gate = GetGate(marker\_circle) ?? throw new Exception("Чё?!"); // Такого не бывает

start\_dist = gate.GetPin(marker\_circle, FindCanvas());

var circle\_pos = start\_dist.GetPos();

marker.StartPoint = marker.EndPoint = circle\_pos;

marker.IsVisible = true;

marker\_mode = mode;

break;

case 8:

if (item is not Line @join) break;

JoinedItems.arrow\_to\_join.TryGetValue(@join, out var @join2);

if (@join2 == null) break;

var dist\_a = @join.StartPoint.Hypot(pos);

var dist\_b = @join.EndPoint.Hypot(pos);

join\_start = dist\_a > dist\_b;

old\_join = @join;

marker.StartPoint = join\_start ? @join.StartPoint : pos;

marker.EndPoint = join\_start ? pos : @join.EndPoint;

marker\_mode = CalcMode(join\_start ? @join2.A.tag : @join2.B.tag);

marker.IsVisible = true;

@join.IsVisible = false;

break;

}

Move(item, pos);

}

public Canvas? FindCanvas() {

foreach (var item in items) {

var p = item.GetSelf().Parent;

if (p is Canvas @canv) return @canv;

}

return null;

}

public void FixItem(ref Control res, Point pos, IEnumerable<ILogical> items) {

foreach (var logic in items) {

var item = (Control) logic;

var tb = item.TransformedBounds;

if (tb != null && tb.Value.Bounds.TransformToAABB(tb.Value.Transform).Contains(pos) && (string?) item.Tag != "Join") res = item;

FixItem(ref res, pos, item.GetLogicalChildren());

}

}

public void Move(Control item, Point pos)

{

if (mode == 5 || mode == 6 || mode == 7 || mode == 8)

{

var canv = FindCanvas();

if (canv != null)

{

var tb = canv.TransformedBounds;

if (tb != null)

{

item = new Canvas() { Tag = "Scene" };

var bounds = tb.Value.Bounds.TransformToAABB(tb.Value.Transform);

FixItem(ref item, pos + bounds.TopLeft, canv.Children);

}

}

}

string[] mods = new[] { "In", "Out", "IO" };

var tag = (string?)item.Tag;

var from\_point = -1;

var to\_point = -1;

if (start\_dist != null) from\_point = items.IndexOf(start\_dist.parent);

if (GetGate(item) != null) to\_point = items.IndexOf(GetGate(item));

//Log.Write("Blya: " + from\_point + " " + to\_point);

if (IsMode(item, mods) && item is Ellipse @ellipse

&& !(marker\_mode == 5 && tag == "In" || marker\_mode == 6 && tag == "Out") && (from\_point != to\_point))

{

if (marker\_circle != null && marker\_circle != @ellipse)

{

marker\_circle.Fill = new SolidColorBrush(Color.Parse("#0000"));

marker\_circle.Stroke = Brushes.Gray;

}

marker\_circle = @ellipse;

@ellipse.Fill = Brushes.Lime;

@ellipse.Stroke = Brushes.Green;

}

else if (marker\_circle != null)

{

marker\_circle.Fill = new SolidColorBrush(Color.Parse("#0000"));

marker\_circle.Stroke = Brushes.Gray;

marker\_circle = null;

}

if (mode == 8) delete\_join = (string?)item.Tag == "Deleter";

var delta = pos - moved\_pos;

if (delta.X == 0 && delta.Y == 0) return;

if (Math.Pow(delta.X, 2) + Math.Pow(delta.Y, 2) > 9) tapped = false;

switch (mode)

{

case 2:

if (moved\_item == null) break;

var new\_pos = item\_old\_pos + delta;

moved\_item.Move(new\_pos);

break;

case 3:

if (moved\_item == null) break;

var new\_size = item\_old\_size + new Size(delta.X, delta.Y);

moved\_item.Resize(new\_size, false);

break;

case 5 or 6 or 7:

var end\_pos = marker\_circle == null ? pos : marker\_circle.Center(FindCanvas());

marker.EndPoint = end\_pos;

break;

case 8:

if (old\_join == null) break;

var p = marker\_circle == null ? pos : marker\_circle.Center(FindCanvas());

if (join\_start) marker.EndPoint = p;

else marker.StartPoint = p;

break;

}

}

public bool tapped = false; // Обрабатывается после Release

public Point tap\_pos; // Обрабатывается после Release

public Line? new\_join; // Обрабатывается после Release

public int Release(Control item, Point pos)

{

Move(item, pos);

switch (mode)

{

case 5 or 6 or 7:

if (start\_dist == null) break;

if (marker\_circle != null)

{

var gate = GetGate(marker\_circle) ?? throw new Exception("Чё?!");

var end\_dist = gate.GetPin(marker\_circle, FindCanvas());

var newy = new JoinedItems(start\_dist, end\_dist);

new\_join = newy.line;

}

start\_dist = null;

marker.IsVisible = false;

marker\_mode = 0;

break;

case 8:

if (old\_join == null) break;

JoinedItems.arrow\_to\_join.TryGetValue(old\_join, out var @join);

if (marker\_circle != null && @join != null)

{

var gate = GetGate(marker\_circle) ?? throw new Exception("Чё?!");

var p = gate.GetPin(marker\_circle, FindCanvas());

@join.Delete();

var newy = join\_start ? new JoinedItems(@join.A, p) : new JoinedItems(p, @join.B);

new\_join = newy.line;

}

else old\_join.IsVisible = true;

marker.IsVisible = false;

marker\_mode = 0;

old\_join = null;

if (delete\_join) @join?.Delete();

delete\_join = false;

break;

}

if (tapped) Tapped(item, pos);

int res\_mode = mode;

mode = 0;

return res\_mode;

}

private void Tapped(Control item, Point pos) {

tap\_pos = pos;

if (mode == 4 && moved\_item != null) RemoveItem(moved\_item);

}

public void WheelMove(Control item, double move) {

// Log.Write("WheelMoved: " + item.GetType().Name + " delta: " + (move > 0 ? 1 : -1));

}

/\*

\* Экспорт и импорт

\*/

public readonly FileHandler filer = new();

public void Export(Scheme current\_scheme) {

var arr = items.Select(x => x.Export()).ToArray();

Dictionary<IGate, int> item\_to\_num = new();

int n = 0;

foreach (var item in items) item\_to\_num.Add(item, n++);

List<object[]> joins = new();

foreach (var item in items) joins.Add(item.ExportJoins(item\_to\_num));

bool[] states = sim.Export();

try { current\_scheme.Update(arr, joins.ToArray(), states); }

catch (Exception e) { Log.Write("Save error:\n" + e); }

Log.Write("Items: " + Utils.Obj2json(arr));

Log.Write("Joins: " + Utils.Obj2json(joins));

Log.Write("States: " + Utils.Obj2json(states));

}

public void ImportScheme(Scheme current\_scheme, Canvas canv) {

sim.lock\_sim = true;

RemoveAll();

List<IGate> list = new();

foreach (var item in current\_scheme.items) {

if (item is not Dictionary<string, object> @dict) { Log.Write("Не верный тип элемента: " + item); continue; }

if (!@dict.TryGetValue("id", out var @value)) { Log.Write("id элемента не обнаружен"); continue; }

if (@value is not int @id) { Log.Write("Неверный тип id: " + @value); continue; }

var newy = CreateItem(@id);

newy.Import(@dict);

AddItem(newy);

canv.Children.Add(newy.GetSelf());

list.Add(newy);

}

var items\_arr = list.ToArray();

List<JoinedItems> joinz = new();

foreach (var obj in current\_scheme.joins) {

if (obj is not List<object> @join) { Log.Write("Одно из соединений не того типа: " + obj); continue; }

if (@join.Count != 6 ||

@join[0] is not int @num\_a || @join[1] is not int @pin\_a || @join[2] is not string @tag\_a ||

@join[3] is not int @num\_b || @join[4] is not int @pin\_b || @join[5] is not string @tag\_b) { Log.Write("Содержимое списка соединения ошибочно"); continue; }

var newy = new JoinedItems(new(items\_arr[@num\_a], @pin\_a, canv, tag\_a), new(items\_arr[@num\_b], @pin\_b, canv, tag\_b));

canv.Children.Add(newy.line);

joinz.Add(newy);

}

sim.Import(current\_scheme.states);

sim.lock\_sim = false;

Task.Run(async () => {

await Task.Delay(50);

await Dispatcher.UIThread.InvokeAsync(() => {

foreach (var join in joinz) join.Update();

});

});

}

}

}

**Models/project.cs**

using System;

using System.Collections.Generic;

using System.Linq;

namespace LogicSimulator.Models {

public class Project: IComparable {

public string Name { get; private set; }

public long Created;

public long Modified;

public List<Scheme> schemes = new();

public List<string> scheme\_files = new();

public string FileName { get; }

public Project() { // Новый проект

Name = "Новый проект";

Created = Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();

FileName = FileHandler.GetProjectFileName();

CreateScheme();

}

public Project(string fileName, object data) { // Импорт

FileName = fileName;

if (data is not Dictionary<string, object> dict) throw new Exception("Ожидался словарь в корне проекта");

if (!dict.TryGetValue("name", out var value)) throw new Exception("В проекте нет имени");

if (value is not string name) throw new Exception("Тип имени проекта - не строка");

Name = name;

if (!dict.TryGetValue("created", out var value2)) throw new Exception("В проекте нет времени создания");

if (value2 is not int create\_t) throw new Exception("Время создания проекта - не строка");

Created = create\_t;

if (!dict.TryGetValue("modified", out var value3)) throw new Exception("В проекте нет времени изменения");

if (value3 is not int mod\_t) throw new Exception("Время изменения проекта - не строка");

Modified = mod\_t;

if (!dict.TryGetValue("schemes", out var value4)) throw new Exception("В проекте нет списка схем");

if (value4 is not List<object> arr) throw new Exception("Списко схем проекта - не массив строк");

foreach (var file in arr) {

if (file is not string str) throw new Exception("Одно из файловых имёт списка схем проекта - не строка");

scheme\_files.Add(str);

}

}

public Scheme CreateScheme() {

var scheme = new Scheme(this);

schemes.Add(scheme);

scheme.Save();

scheme\_files.Add(scheme.FileName);

Save();

return scheme;

}

bool loaded = false;

private void LoadSchemes() {

if (loaded) return;

foreach (var fileName in scheme\_files) {

var scheme = FileHandler.LoadScheme(this, fileName);

if (scheme != null) schemes.Add(scheme);

}

loaded = true;

}

public Scheme GetFirstCheme() {

LoadSchemes();

return schemes[0];

}

public object Export() {

return new Dictionary<string, object> {

["name"] = Name,

["created"] = Created,

["modified"] = Modified,

["schemes"] = schemes.Select(x => x.FileName).ToArray(),

};

}

public void Save() => FileHandler.SaveProject(this);

public int CompareTo(object? obj) {

if (obj is not Project proj) throw new ArgumentNullException(nameof(obj));

return (int)(proj.Modified - Modified);

}

public override string ToString() {

return Name + "\nИзменён: " + Modified.UnixTimeStampToString() + "\nСоздан: " + Created.UnixTimeStampToString();

}

internal void ChangeName(string name) {

Name = name;

Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();

Save();

}

}

}

**Models/simulator.cs**

using LogicSimulator.ViewModels;

using LogicSimulator.Views.Shapes;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace LogicSimulator.Models {

public class Meta {

public IGate? item;

public int[] ins;

public int[] outs;

public bool[] i\_buf;

public bool[] o\_buf;

public Meta(IGate item, int out\_id) {

this.item = item;

ins = Enumerable.Repeat(0, item.CountIns).ToArray();

outs = Enumerable.Range(out\_id, item.CountOuts).ToArray();

i\_buf = Enumerable.Repeat(false, item.CountIns).ToArray();

o\_buf = Enumerable.Repeat(false, item.CountOuts).ToArray();

}

public void Print() {

Log.Write("Элемент: " + item + " | Ins: " + Utils.Obj2json(ins) + " | Outs: " + Utils.Obj2json(outs));

}

}

public class Simulator {

public bool lock\_sim = false;

public Simulator() {

var task = Task.Run(async () => {

for (;;) {

await Task.Delay(1000 / 60);

if (lock\_sim) continue;

try { Tick(); }

catch (Exception e) { Log.Write("Logical crush: " + e); continue; }

}

});

}

List<bool> outs = new() { false };

List<bool> outs2 = new() { false };

readonly List<Meta> items = new();

readonly Dictionary<IGate, Meta> ids = new();

public void AddItem(IGate item) {

lock\_sim = true;

int out\_id = outs.Count;

for (int i = 0; i < item.CountOuts; i++) {

outs.Add(false);

outs2.Add(false);

}

// int id = items.Count;

Meta meta = new(item, out\_id);

items.Add(meta);

ids.Add(item, meta);

lock\_sim = false;

// meta.Print();

}

public void RemoveItem(IGate item) {

lock\_sim = true;

Meta meta = ids[item];

meta.item = null;

foreach (var i in Enumerable.Range(0, meta.outs.Length)) meta.outs[i] = 0;

lock\_sim = false;

}

private void Tick() {

foreach (var meta in items) {

var item = meta.item;

if (item == null) continue;

item.LogicUpdate(ids, meta);

int[] i\_n = meta.ins, o\_n = meta.outs;

bool[] ib = meta.i\_buf, ob = meta.o\_buf;

for (int i = 0; i < ib.Length; i++) ib[i] = outs[i\_n[i]];

item.Brain(ref ib, ref ob);

for (int i = 0; i < ob.Length; i++) {

bool res = ob[i];

outs2[o\_n[i]] = res;

item.SetJoinColor(i, res);

}

}

(outs2, outs) = (outs, outs2);

}

public bool[] Export() => outs.ToArray();

public void Import(bool[] state) {

if (state.Length == 0) state = new bool[] { false };

outs = state.ToList();

outs2 = Enumerable.Repeat(false, state.Length).ToList();

}

}

}

**Models/fileHandler.cs**

using LogicSimulator.ViewModels;

using System;

using System.Collections.Generic;

using System.Data;

using System.IO;

namespace LogicSimulator.Models {

public class FileHandler {

readonly static string dir = "../../../../storage/";

readonly List<Project> projects = new();

public FileHandler() {

if (!Directory.Exists(dir)) Directory.CreateDirectory(dir);

foreach (var fullname in Directory.EnumerateFiles(dir)) {

var name = fullname.Split("/")[^1];

if (name.StartsWith("proj\_")) LoadProject(name);

}

}

public static string GetProjectFileName() {

int n = 0;

while (true) {

string name = "proj\_" + ++n + ".json";

if (!File.Exists(dir + name)) return name;

}

}

public static string GetSchemeFileName() {

int n = 0;

while (true) {

string name = "scheme\_" + ++n + ".yaml";

if (!File.Exists(dir + name)) return name;

}

}

public Project CreateProject() {

var proj = new Project();

projects.Add(proj);

return proj;

}

private Project? LoadProject(string fileName) {

try {

var obj = Utils.Json2obj(File.ReadAllText(dir + fileName)) ?? throw new DataException("Не верная структура JSON-файла проекта!");

var proj = new Project(fileName, obj);

projects.Add(proj);

return proj;

} catch (Exception e) { Log.Write("Неудачная попытка загрузить проект:\n" + e); }

return null;

}

public static Scheme? LoadScheme(Project parent, string fileName) {

try {

var obj = Utils.Yaml2obj(File.ReadAllText(dir + fileName)) ?? throw new DataException("Не верная структура YAML-файла схемы!");

var scheme = new Scheme(parent, fileName, obj);

return scheme;

} catch (Exception e) { Log.Write("Неудачная попытка загрузить схему:\n" + e); }

return null;

}

public static void SaveProject(Project proj) {

var data = Utils.Obj2json(proj.Export());

File.WriteAllText(dir + proj.FileName, data);

}

public static void SaveScheme(Scheme scheme) {

var data = Utils.Obj2yaml(scheme.Export());

File.WriteAllText(dir + scheme.FileName, data);

}

public Project[] GetSortedProjects() {

projects.Sort();

return projects.ToArray();

}

}

}

**Models/scheme.cs**

using System;

using System.Collections.Generic;

using System.Linq;

namespace LogicSimulator.Models {

public class Scheme {

public string Name { get; set; }

public long Created;

public long Modified;

public object[] items;

public object[] joins;

public bool[] states;

public string FileName { get; }

private readonly Project parent;

public Scheme(Project p) { // Новая схема

Created = Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();

Name = "Newy";

items = joins = Array.Empty<object>();

states = Array.Empty<bool>();

FileName = FileHandler.GetSchemeFileName();

parent = p;

}

public Scheme(Project p, string fileName, object data) { // Импорт

FileName = fileName;

parent = p;

if (data is not Dictionary<string, object> dict) throw new Exception("Ожидался словарь в корне схемы");

if (!dict.TryGetValue("name", out var value)) throw new Exception("В схеме нет имени");

if (value is not string name) throw new Exception("Тип имени схемы - не строка");

Name = name;

if (!dict.TryGetValue("created", out var value2)) throw new Exception("В схеме нет времени создания");

if (value2 is not int create\_t) throw new Exception("Время создания схемы - не строка");

Created = create\_t;

if (!dict.TryGetValue("modified", out var value3)) throw new Exception("В схеме нет времени изменения");

if (value3 is not int mod\_t) throw new Exception("Время изменения схемы - не строка");

Modified = mod\_t;

if (!dict.TryGetValue("items", out var value4)) throw new Exception("В схеме нет списка элементов");

if (value4 is not List<object> arr) throw new Exception("Список элементов схемы - не массив объектов");

items = arr.ToArray();

if (!dict.TryGetValue("joins", out var value5)) throw new Exception("В схеме нет списка соединений");

if (value5 is not List<object> arr2) throw new Exception("Список соединений схемы - не массив объектов");

joins = arr2.ToArray();

if (!dict.TryGetValue("states", out var value6)) throw new Exception("В схеме нет списка состояний");

if (value6 is not List<object> arr3) throw new Exception("Список состояний схемы - не массив bool");

states = arr3.Select(x => (bool) x).ToArray();

}

public void Update(object[] items, object[] joins, bool[] states) {

this.items = items;

this.joins = joins;

this.states = states;

Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();

Update();

}

public object Export() {

return new Dictionary<string, object> {

["name"] = Name,

["created"] = Created,

["modified"] = Modified,

["items"] = items,

["joins"] = joins,

["states"] = states,

};

}

public void Save() => FileHandler.SaveScheme(this);

public void Update() {

Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();

parent.Modified = Modified;

parent.Save();

Save();

}

public override string ToString() => Name;

internal void ChangeName(string name) {

Name = name;

Update();

}

}

}

**Models/joinedItems.cs**

using Avalonia.Controls.Shapes;

using Avalonia.Media;

using System.Collections.Generic;

namespace LogicSimulator.Models {

public class JoinedItems {

public static readonly Dictionary<Line, JoinedItems> arrow\_to\_join = new();

public JoinedItems(Distantor a, Distantor b) {

A = a; B = b; Update();

a.parent.AddJoin(this);

b.parent.AddJoin(this);

arrow\_to\_join[line] = this;

}

public Distantor A { get; set; }

public Distantor B { get; set; }

public Line line = new() { Tag = "Join", ZIndex = 2, Stroke = Brushes.DarkGray, StrokeThickness = 3 };

public void Update() {

line.StartPoint = A.GetPos();

line.EndPoint = B.GetPos();

}

public void Delete() {

arrow\_to\_join.Remove(line);

line.Remove();

A.parent.RemoveJoin(this);

B.parent.RemoveJoin(this);

}

}

}

**ViewModels/LauncherWindowViewModel.cs**

using Avalonia.Controls.Presenters;

using Avalonia.Controls;

using ReactiveUI;

using System.Reactive;

using LogicSimulator.Views;

using LogicSimulator.Models;

namespace LogicSimulator.ViewModels {

public class LauncherWindowViewModel: ViewModelBase {

Window? me;

private static readonly MainWindow mw = new();

public LauncherWindowViewModel() {

Create = ReactiveCommand.Create<Unit, Unit>(\_ => { FuncCreate(); return new Unit(); });

Exit = ReactiveCommand.Create<Unit, Unit>(\_ => { FuncExit(); return new Unit(); });

}

public void AddWindow(Window lw) => me = lw;

void FuncCreate() {

var newy = map.filer.CreateProject();

current\_proj = newy;

current\_scheme = current\_proj.GetFirstCheme();

mw.Show();

mw.Update();

me?.Close();

}

void FuncExit() {

me?.Close();

}

public ReactiveCommand<Unit, Unit> Create { get; }

public ReactiveCommand<Unit, Unit> Exit { get; }

public static Project[] ProjectList { get => map.filer.GetSortedProjects(); }

public void DTapped(object? sender, Avalonia.Interactivity.RoutedEventArgs e) {

var src = (Control?) e.Source;

if (src is ContentPresenter cp && cp.Child is Border bord) src = bord;

if (src is Border bord2 && bord2.Child is TextBlock tb2) src = tb2;

if (src is not TextBlock tb || tb.Tag is not Project proj) return;

current\_proj = proj;

current\_scheme = current\_proj.GetFirstCheme();

mw.Show();

mw.Update();

me?.Close();

}

}

}

**ViewModels/MainWindowViewModel.cs**

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.Presenters;

using Avalonia.Input;

using LogicSimulator.Models;

using LogicSimulator.Views;

using LogicSimulator.Views.Shapes;

using ReactiveUI;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.IO;

using System.Reactive;

namespace LogicSimulator.ViewModels {

public class Log {

static readonly List<string> logs = new();

static readonly string path = "../../../Log.txt";

static bool first = true;

static readonly bool use\_file = false;

public static MainWindowViewModel? Mwvm { private get; set; }

public static void Write(string message, bool without\_update = false) {

if (!without\_update) {

foreach (var mess in message.Split('\n')) logs.Add(mess);

while (logs.Count > 50) logs.RemoveAt(0);

if (Mwvm != null) Mwvm.Logg = string.Join('\n', logs);

}

if (use\_file) {

if (first) File.WriteAllText(path, message + "\n");

else File.AppendAllText(path, message + "\n");

first = false;

}

}

}

public class MainWindowViewModel: ViewModelBase, INotifyPropertyChanged {

private string log = "";

public string Logg { get => log; set {

if (log == value) return;

log = value;

PropertyChanged?.Invoke(this, new(nameof(Logg)));

} }

public MainWindowViewModel() {

Log.Mwvm = this;

Comm = ReactiveCommand.Create<string, Unit>(n => { FuncComm(n); return new Unit(); });

}

private Window? mw;

private Canvas? canv;

public void AddWindow(Window window) {

var canv = window.Find<Canvas>("Canvas");

mw = window;

this.canv = canv;

if (canv == null) return; // Такого не бывает

canv.Children.Add(map.Marker);

var panel = (Panel?) canv.Parent;

if (panel == null) return; // Такого не бывает

panel.PointerPressed += (object? sender, PointerPressedEventArgs e) => {

if (e.Source != null && e.Source is Control @control) map.Press(@control, e.GetCurrentPoint(canv).Position);

};

panel.PointerMoved += (object? sender, PointerEventArgs e) => {

if (e.Source != null && e.Source is Control @control) map.Move(@control, e.GetCurrentPoint(canv).Position);

};

panel.PointerReleased += (object? sender, PointerReleasedEventArgs e) => {

if (e.Source != null && e.Source is Control @control) {

int mode = map.Release(@control, e.GetCurrentPoint(canv).Position);

bool tap = map.tapped;

if (tap && mode == 1) {

var pos = map.tap\_pos;

if (canv == null) return; // Такого не бывает

var newy = map.GenSelectedItem();

var size = newy.GetSize() / 2;

newy.Move(pos - new Point(size.Width, size.Height));

canv.Children.Add(newy.GetSelf());

map.AddItem(newy);

}

if (map.new\_join != null) {

canv.Children.Add(map.new\_join);

map.new\_join = null;

}

}

};

panel.PointerWheelChanged += (object? sender, PointerWheelEventArgs e) => {

if (e.Source != null && e.Source is Control @control) map.WheelMove(@control, e.Delta.Y);

};

}

public static IGate[] ItemTypes { get => map.item\_types; }

public static int SelectedItem { get => map.SelectedItem; set => map.SelectedItem = value; }

/\*

\* Обработка той самой панели со схемами проекта

\*/

Border? cur\_border;

TextBlock? old\_b\_child;

object? old\_b\_child\_tag;

string? prev\_scheme\_name;

public static string ProjName { get => current\_proj == null ? "???" : current\_proj.Name; }

public static List<Scheme> Schemes { get => current\_proj == null ? new() : current\_proj.schemes; }

public void DTapped(object? sender, Avalonia.Interactivity.RoutedEventArgs e) {

var src = (Control?) e.Source;

if (src is ContentPresenter cp && cp.Child is Border bord) src = bord;

if (src is Border bord2 && bord2.Child is TextBlock tb2) src = tb2;

if (src is not TextBlock tb) return;

var p = tb.Parent;

if (p == null || p is not Border b) return;

if (cur\_border != null && old\_b\_child != null) cur\_border.Child = old\_b\_child;

cur\_border = b;

old\_b\_child = tb;

old\_b\_child\_tag = tb.Tag;

prev\_scheme\_name = tb.Text;

var newy = new TextBox { Text = tb.Text };

// Log.Write("Tag: " + tb.Tag);

b.Child = newy;

//Log.Write("Tag: " + tb.Tag); // КААААК?!?!?!? Почему пропажа предка удаляет Tag?!

newy.KeyUp += (object? sender, KeyEventArgs e) => {

if (e.Key != Key.Return) return;

if (newy.Text != prev\_scheme\_name) {

// tb.Text = newy.Text;

if ((string?) tb.Tag == "p\_name") current\_proj?.ChangeName(newy.Text);

else if (old\_b\_child\_tag is Scheme scheme) scheme.ChangeName(newy.Text);

}

b.Child = tb;

cur\_border = null; old\_b\_child = null;

};

}

#pragma warning disable CS0108

public event PropertyChangedEventHandler? PropertyChanged;

#pragma warning restore CS0108

public void Update() {

Log.Write("Текущий проект:\n" + current\_proj);

if (current\_scheme == null || canv == null) throw new Exception("Такого не бывает");

map.ImportScheme(current\_scheme, canv);

PropertyChanged?.Invoke(this, new(nameof(ProjName)));

PropertyChanged?.Invoke(this, new(nameof(Schemes)));

}

/\*

\* Кнопочки!

\*/

public void FuncComm(string Comm) {

Log.Write("Comm: " + Comm);

switch (Comm) {

case "Create":

break;

case "Open":

new LauncherWindow().Show();

mw?.Hide();

break;

case "Save":

if (current\_scheme != null) map.Export(current\_scheme);

break;

case "Exit":

mw?.Close();

break;

}

}

public ReactiveCommand<string, Unit> Comm { get; }

}

}

**Models/ViewModelBase.cs**

using LogicSimulator.Models;

using ReactiveUI;

namespace LogicSimulator.ViewModels {

public class ViewModelBase: ReactiveObject {

protected readonly static Mapper map = new();

protected static Project? current\_proj;

protected static Scheme? current\_scheme;

}

}

**Views/Shapes/GateBase.cs**

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.Shapes;

using Avalonia.Media;

using Avalonia.Threading;

using LogicSimulator.Models;

using LogicSimulator.ViewModels;

using System;

using System.Collections.Generic;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public abstract class GateBase: UserControl {

public abstract int CountIns { get; }

public abstract int CountOuts { get; }

public abstract UserControl GetSelf();

protected abstract IGate GetSelfI { get; }

protected abstract void Init();

protected Ellipse[] pins;

public GateBase() {

Init();

int count = CountIns + CountOuts;

List<Ellipse> list = new();

foreach (var logic in LogicalChildren[0].LogicalChildren)

if (logic is Ellipse @ellipse) list.Add(@ellipse);

if (list.Count != count) throw new Exception("Чё?!"); // У этой фигуры всегда count пинов

pins = list.ToArray();

joins = new JoinedItems?[count];

}

public void Move(Point pos) {

Margin = new(pos.X, pos.Y, 0, 0);

UpdateJoins(false);

}

public void Resize(Size size, bool global) {

double limit = (9 + 32) \* 2;

width = size.Width.Max(limit / 3 \* (CountIns == 0 || CountOuts == 0 ? 2.25 : 3));

height = size.Height.Max(limit / 3 \* (1.5 + 0.75 \* CountIns.Max(CountOuts)));

RecalcSizes();

UpdateJoins(global);

}

public Point GetPos() => new(Margin.Left, Margin.Top);

public Size GetSize() => new(Width, Height);

public Size GetBodySize() => new(width, height);

/\*

\* Обработка размеров внутренностей

\*/

protected readonly double base\_size = 25;

protected double width = 30 \* 3;

protected double height = 30 \* 3;

public double BaseSize => base\_size;

public double BaseFraction => base\_size / 40;

public double EllipseSize => BaseFraction \* 30;

public Thickness BodyStrokeSize => new(BaseFraction \* 3);

public double EllipseStrokeSize => BaseFraction \* 5;

public double PinStrokeSize => BaseFraction \* 6;

public Thickness BodyMargin => new(base\_size, 0, 0, 0);

public double BodyWidth => width;

public double BodyHeight => height;

public CornerRadius BodyRadius => new(width.Min(height) / 3 + BodyStrokeSize.Top);

public double UC\_Width => base\_size \* 2 + width;

public double UC\_Height => height;

public double FontSizze => BodyRadius.TopLeft / 1.3;

public Thickness[] ImageMargins {

get {

double R = BodyRadius.BottomLeft;

double num = R - R / Math.Sqrt(2);

return new Thickness[] {

new(0, 0, num, num), // Картинка с удалителем

new(num, 0, 0, num), // Картинка с переместителем

};

} }

public abstract Point[][] PinPoints { get; }

public Thickness[] EllipseMargins { get {

Point[][] pins = PinPoints;

double R2 = EllipseSize / 2;

double X = UC\_Width - EllipseSize;

int n = 0;

List<Thickness> list = new();

foreach (var pin\_line in pins)

list.Add(new(n++ < CountIns ? 0 : X, pin\_line[0].Y - R2, 0, 0));

return list.ToArray();

} }

#pragma warning disable CS0108

public event PropertyChangedEventHandler? PropertyChanged;

#pragma warning restore CS0108

protected void RecalcSizes() {

// Log.Write("Size: " + width + " " + height);

PropertyChanged?.Invoke(this, new(nameof(EllipseSize)));

PropertyChanged?.Invoke(this, new(nameof(BodyStrokeSize)));

PropertyChanged?.Invoke(this, new(nameof(EllipseStrokeSize)));

PropertyChanged?.Invoke(this, new(nameof(PinStrokeSize)));

PropertyChanged?.Invoke(this, new(nameof(BodyMargin)));

PropertyChanged?.Invoke(this, new(nameof(BodyWidth)));

PropertyChanged?.Invoke(this, new(nameof(BodyHeight)));

PropertyChanged?.Invoke(this, new(nameof(BodyRadius)));

PropertyChanged?.Invoke(this, new(nameof(EllipseMargins)));

PropertyChanged?.Invoke(this, new(nameof(PinPoints)));

PropertyChanged?.Invoke(this, new(nameof(UC\_Width)));

PropertyChanged?.Invoke(this, new(nameof(UC\_Height)));

PropertyChanged?.Invoke(this, new(nameof(FontSizze)));

PropertyChanged?.Invoke(this, new(nameof(ImageMargins)));

PropertyChanged?.Invoke(this, new("ButtonSize"));

}

/\*

\* Обработка соединений

\*/

protected JoinedItems?[] joins;

public void AddJoin(JoinedItems join) {

if (join.A.parent == this) {

int n = join.A.num;

joins[n]?.Delete();

joins[n] = join;

}

if (join.B.parent == this) {

int n = join.B.num;

joins[n]?.Delete();

joins[n] = join;

}

skip\_upd = false;

}

public void RemoveJoin(JoinedItems join) {

if (join.A.parent == this) joins[join.A.num] = null;

if (join.B.parent == this) joins[join.B.num] = null;

skip\_upd = false;

}

public void UpdateJoins(bool global) {

foreach (var join in joins)

if (join != null && (!global || join.A.parent == this)) join.Update();

}

public void ClearJoins() {

foreach (var join in joins) join?.Delete();

}

public void SetJoinColor(int o\_num, bool value) {

var join = joins[o\_num + CountIns];

if (join != null)

Dispatcher.UIThread.InvokeAsync(() => {

join.line.Stroke = value ? Brushes.Lime : Brushes.DarkGray;

});

}

/\*

\* Обработка пинов

\*/

public Distantor GetPin(Ellipse finded, Visual? ref\_point) {

int n = 0;

foreach (var pin in pins) {

if (pin == finded) return new(GetSelfI, n, ref\_point, (string?) finded.Tag ?? "");

n++;

}

throw new Exception("Так не бывает");

}

public Point GetPinPos(int n, Visual? ref\_point) {

var pin = pins[n];

return pin.Center(ref\_point);

}

/\*

\* Мозги

\*/

bool skip\_upd = true;

public void LogicUpdate(Dictionary<IGate, Meta> ids, Meta me) {

if (skip\_upd) return;

skip\_upd = true;

int ins = CountIns;

for (int i = 0; i < ins; i++) {

var join = joins[i];

if (join == null) { me.ins[i] = 0; continue; }

if (join.A.parent == this) {

var item = join.B;

if (item.tag == "Out" || item.tag == "IO") {

var p = item.parent;

Meta meta = ids[p];

me.ins[i] = meta.outs[item.num - p.CountIns];

}

}

if (join.B.parent == this) {

var item = join.A;

if (item.tag == "Out" || item.tag == "IO") {

var p = item.parent;

Meta meta = ids[p];

me.ins[i] = meta.outs[item.num - p.CountIns];

}

}

}

}

/\*

\* Экспорт

\*/

public abstract int TypeId { get; }

public virtual object Export() {

return new Dictionary<string, object> {

["id"] = TypeId,

["pos"] = GetPos(),

["size"] = GetBodySize()

};

}

public List<object[]> ExportJoins(Dictionary<IGate, int> to\_num) {

List<object[]> res = new();

int n = 0, ins = CountIns;

foreach (var join in joins) {

if (++n > ins) break;

if (join == null) continue;

Distantor a = join.A, b = join.B;

res.Add(new object[] {

to\_num[a.parent], a.num, a.tag,

to\_num[b.parent], b.num, b.tag,

});

}

return res;

}

public virtual void Import(Dictionary<string, object> dict) {

if (!@dict.TryGetValue("pos", out var @value)) { Log.Write("pos-запись элемента не обнаружен"); return; }

if (@value is not Point @pos) { Log.Write("Неверный тип pos-записи элемента: " + @value); return; }

Move(@pos);

if (!@dict.TryGetValue("size", out var @value2)) { Log.Write("size-запись элемента не обнаружен"); return; }

if (@value2 is not Size @size) { Log.Write("Неверный тип size-записи элемента: " + @value2); return; }

Resize(@size, false);

}

}

}

**Views/Shapes/GateBase.cs**

using Avalonia;

using Avalonia.Controls;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public partial class Shifrator: GateBase, IGate, INotifyPropertyChanged {

public override int TypeId => 4;

public override int CountIns => 4;

public override int CountOuts => 2;

public override UserControl GetSelf() => this;

protected override IGate GetSelfI => this;

protected override void Init() {

height = 30 \* 4;

InitializeComponent();

DataContext = this;

}

/\*

\* Обработка размеров внутренностей

\*/

public override Point[][] PinPoints { get {

double X = EllipseSize - EllipseStrokeSize / 2;

double X2 = base\_size + width - EllipseStrokeSize / 2;

double R = BodyRadius.TopLeft;

double Y\_s = R, Y\_m = height / 2, Y\_e = height - Y\_s;

double min = EllipseSize + BaseFraction \* 2;

// .1..2..3..4.

double Y = Y\_s + (Y\_e - Y\_s) / 8;

double Y2 = Y\_s + (Y\_e - Y\_s) / 8 \* 3;

double Y3 = Y\_s + (Y\_e - Y\_s) / 8 \* 5;

double Y4 = Y\_s + (Y\_e - Y\_s) / 8 \* 7;

if (Y2 - Y < min) { Y = Y\_m - min / 2 \* 3; Y2 = Y\_m - min / 2; Y3 = Y\_m + min / 2; Y4 = Y\_m + min / 2 \* 3; }

double PinWidth = base\_size - EllipseSize + PinStrokeSize;

return new Point[][] {

new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход

new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход

new Point[] { new(X, Y3), new(X + PinWidth, Y3) }, // Третий вход

new Point[] { new(X, Y4), new(X + PinWidth, Y4) }, // Четвёртый вход

new Point[] { new(X2, Y2), new(X2 + PinWidth, Y2) }, // Первый выход

new Point[] { new(X2, Y3), new(X2 + PinWidth, Y3) }, // Второй выход

};

} }

/\*

\* Мозги

\*/

public void Brain(ref bool[] ins, ref bool[] outs) {

bool a = ins[0], b = ins[1], c = ins[2], d = ins[3];

if (d) {

outs[0] = true;

outs[1] = true;

} else if (c) {

outs[0] = false;

outs[1] = true;

} else if (b) {

outs[0] = true;

outs[1] = false;

} else if (a) {

outs[0] = false;

outs[1] = false;

}

}

}

}

**Views/Shapes/AND\_2.axaml.cs**

using Avalonia;

using Avalonia.Controls;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public partial class AND\_2: GateBase, IGate, INotifyPropertyChanged {

public override int TypeId => 0;

public override int CountIns => 2;

public override int CountOuts => 1;

public override UserControl GetSelf() => this;

protected override IGate GetSelfI => this;

protected override void Init() {

height = 30 \* 3;

InitializeComponent();

DataContext = this;

}

/\*

\* Обработка размеров внутренностей

\*/

public override Point[][] PinPoints { get {

double X = EllipseSize - EllipseStrokeSize / 2;

double X2 = base\_size + width - EllipseStrokeSize / 2;

double R = BodyRadius.TopLeft;

double Y\_s = R, Y\_m = height / 2, Y\_e = height - Y\_s;

double min = EllipseSize + BaseFraction \* 2;

// .1..2.

double Y = Y\_s + (Y\_e - Y\_s) / 4;

double Y2 = Y\_s + (Y\_e - Y\_s) / 4 \* 3;

if (Y2 - Y < min) { Y = Y\_m - min / 2; Y2 = Y\_m + min / 2; }

double PinWidth = base\_size - EllipseSize + PinStrokeSize;

return new Point[][] {

new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход

new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход

new Point[] { new(X2, Y\_m), new(X2 + PinWidth, Y\_m) }, // Единственный выход

};

} }

/\*

\* Мозги

\*/

public void Brain(ref bool[] ins, ref bool[] outs) => outs[0] = ins[0] && ins[1];

}

}

**Views/Shapes/IGate.cs**

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.Shapes;

using LogicSimulator.Models;

using System.Collections.Generic;

namespace LogicSimulator.Views.Shapes {

public interface IGate {

public int CountIns { get; }

public int CountOuts { get; }

public UserControl GetSelf();

public Point GetPos();

public Size GetSize();

public Size GetBodySize();

public void Move(Point pos);

public void Resize(Size size, bool global);

public Distantor GetPin(Ellipse finded, Visual? ref\_point);

public Point GetPinPos(int n, Visual? ref\_point);

public void AddJoin(JoinedItems join);

public void RemoveJoin(JoinedItems join);

public void ClearJoins();

public void SetJoinColor(int o\_num, bool value);

public void Brain(ref bool[] ins, ref bool[] outs);

public void LogicUpdate(Dictionary<IGate, Meta> ids, Meta me);

public int TypeId { get; }

public object Export();

public List<object[]> ExportJoins(Dictionary<IGate, int> to\_num);

public void Import(Dictionary<string, object> dict);

}

}

**Views/Shapes/NOT.axaml.cs**

using Avalonia;

using Avalonia.Controls;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public partial class NOT: GateBase, IGate, INotifyPropertyChanged {

public override int TypeId => 2;

public override int CountIns => 1;

public override int CountOuts => 1;

public override UserControl GetSelf() => this;

protected override IGate GetSelfI => this;

protected override void Init() {

height = 30 \* 2.5;

InitializeComponent();

DataContext = this;

}

/\*

\* Обработка размеров внутренностей

\*/

public override Point[][] PinPoints { get {

double X = EllipseSize - EllipseStrokeSize / 2;

double X2 = base\_size + width - EllipseStrokeSize / 2;

double Y = height / 2;

double PinWidth = base\_size - EllipseSize + PinStrokeSize;

return new Point[][] {

new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Единственный вход

new Point[] { new(X2, Y), new(X2 + PinWidth, Y) }, // Единственный выход

};

} }

/\*

\* Мозги

\*/

public void Brain(ref bool[] ins, ref bool[] outs) => outs[0] = !ins[0];

}

}

**Views/Shapes/OR\_2.axaml.cs**

using Avalonia;

using Avalonia.Controls;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public partial class OR\_2: GateBase, IGate, INotifyPropertyChanged {

public override int TypeId => 1;

public override int CountIns => 2;

public override int CountOuts => 1;

public override UserControl GetSelf() => this;

protected override IGate GetSelfI => this;

protected override void Init() {

height = 30 \* 3;

InitializeComponent();

DataContext = this;

}

/\*

\* Обработка размеров внутренностей

\*/

public override Point[][] PinPoints { get {

double X = EllipseSize - EllipseStrokeSize / 2;

double X2 = base\_size + width - EllipseStrokeSize / 2;

double R = BodyRadius.TopLeft;

double Y\_s = R, Y\_m = height / 2, Y\_e = height - Y\_s;

double min = EllipseSize + BaseFraction \* 2;

// .1..2.

double Y = Y\_s + (Y\_e - Y\_s) / 4;

double Y2 = Y\_s + (Y\_e - Y\_s) / 4 \* 3;

if (Y2 - Y < min) { Y = Y\_m - min / 2; Y2 = Y\_m + min / 2; }

double PinWidth = base\_size - EllipseSize + PinStrokeSize;

return new Point[][] {

new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход

new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход

new Point[] { new(X2, Y\_m), new(X2 + PinWidth, Y\_m) }, // Единственный выход

};

} }

/\*

\* Мозги

\*/

public void Brain(ref bool[] ins, ref bool[] outs) => outs[0] = ins[0] || ins[1];

}

}

**Views/Shapes/switch.axaml.cs**

using Avalonia;

using Avalonia.Controls;

using Avalonia.Input;

using Avalonia.Media;

using LogicSimulator.Models;

using LogicSimulator.ViewModels;

using System;

using System.Collections.Generic;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public partial class Switch: GateBase, IGate, INotifyPropertyChanged {

public override int TypeId => 5;

public override int CountIns => 0;

public override int CountOuts => 1;

public override UserControl GetSelf() => this;

protected override IGate GetSelfI => this;

protected override void Init() {

width = 30 \* 2.5;

height = 30 \* 2.5;

InitializeComponent();

DataContext = this;

}

readonly Border border;

public Switch() : base() {

if (LogicalChildren[0].LogicalChildren[1] is not Border b) throw new Exception("Такого не бывает");

border = b;

}

/\*

\* Обработка размеров внутренностей

\*/

public override Point[][] PinPoints { get {

double X = base\_size + width - EllipseStrokeSize / 2;

double Y = height / 2;

double PinWidth = base\_size - EllipseSize + PinStrokeSize;

return new Point[][] {

new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Единственный выход

};

} }

/\*

\* Мозги

\*/

bool my\_state = false;

Point? press\_pos;

// Данная схема работает гораздо быстрее, чем событие Tapped ;'-} Из-за того, что не обрабатывается дополнительно DoubleTapped, что гасит второй Tapped + некоторые задержки

private static Point GetPos(PointerEventArgs e) {

if (e.Source is not Control src) return new();

while ((string?) src.Tag != "scene" && src.Parent != null) src = (Control) src.Parent;

return e.GetCurrentPoint(src).Position;

}

private void Press(object? sender, PointerPressedEventArgs e) {

if (e.Source == border) press\_pos = GetPos(e);

}

private void Release(object? sender, PointerReleasedEventArgs e) {

if (e.Source != border) return;

if (press\_pos == null || GetPos(e).Hypot((Point) press\_pos) > 5) return;

press\_pos = null;

my\_state = !my\_state;

border.Background = new SolidColorBrush(Color.Parse(my\_state ? "#7d1414" : "#d32f2e"));

}

public void Brain(ref bool[] ins, ref bool[] outs) => outs[0] = my\_state;

/\*

\* Кастомный экспорт и импорт

\*/

public override object Export() {

return new Dictionary<string, object> {

["id"] = TypeId,

["pos"] = GetPos(),

["size"] = GetBodySize(),

["state"] = my\_state

};

}

public override void Import(Dictionary<string, object> dict) {

if (!@dict.TryGetValue("pos", out var @value)) { Log.Write("pos-запись элемента не обнаружен"); return; }

if (@value is not Point @pos) { Log.Write("Неверный тип pos-записи элемента: " + @value); return; }

Move(@pos);

if (!@dict.TryGetValue("size", out var @value2)) { Log.Write("size-запись элемента не обнаружен"); return; }

if (@value2 is not Size @size) { Log.Write("Неверный тип size-записи элемента: " + @value2); return; }

Resize(@size, false);

if (!@dict.TryGetValue("state", out var @value3)) { Log.Write("state-запись элемента не обнаружен"); return; }

if (@value3 is not bool @state) { Log.Write("Неверный тип state-записи элемента: " + @value3); return; }

my\_state = @state;

if (my\_state) border.Background = new SolidColorBrush(Color.Parse("#7d1414"));

}

}

}

**Views/Shapes/xor2.axaml.cs**

using Avalonia;

using Avalonia.Controls;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public partial class XOR\_2: GateBase, IGate, INotifyPropertyChanged {

public override int TypeId => 3;

public override int CountIns => 2;

public override int CountOuts => 1;

public override UserControl GetSelf() => this;

protected override IGate GetSelfI => this;

protected override void Init() {

height = 30 \* 3;

InitializeComponent();

DataContext = this;

}

/\*

\* Обработка размеров внутренностей

\*/

public override Point[][] PinPoints { get {

double X = EllipseSize - EllipseStrokeSize / 2;

double X2 = base\_size + width - EllipseStrokeSize / 2;

double R = BodyRadius.TopLeft;

double Y\_s = R, Y\_m = height / 2, Y\_e = height - Y\_s;

double min = EllipseSize + BaseFraction \* 2;

// .1..2.

double Y = Y\_s + (Y\_e - Y\_s) / 4;

double Y2 = Y\_s + (Y\_e - Y\_s) / 4 \* 3;

if (Y2 - Y < min) { Y = Y\_m - min / 2; Y2 = Y\_m + min / 2; }

double PinWidth = base\_size - EllipseSize + PinStrokeSize;

return new Point[][] {

new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход

new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход

new Point[] { new(X2, Y\_m), new(X2 + PinWidth, Y\_m) }, // Единственный выход

};

} }

/\*

\* Мозги

\*/

public void Brain(ref bool[] ins, ref bool[] outs) => outs[0] = ins[0] ^ ins[1];

}

}

**Views/Shapes/shifrator.axaml.cs**

using Avalonia;

using Avalonia.Controls;

using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {

public partial class Shifrator: GateBase, IGate, INotifyPropertyChanged {

public override int TypeId => 4;

public override int CountIns => 4;

public override int CountOuts => 2;

public override UserControl GetSelf() => this;

protected override IGate GetSelfI => this;

protected override void Init() {

height = 30 \* 4;

InitializeComponent();

DataContext = this;

}

/\*

\* Обработка размеров внутренностей

\*/

public override Point[][] PinPoints { get {

double X = EllipseSize - EllipseStrokeSize / 2;

double X2 = base\_size + width - EllipseStrokeSize / 2;

double R = BodyRadius.TopLeft;

double Y\_s = R, Y\_m = height / 2, Y\_e = height - Y\_s;

double min = EllipseSize + BaseFraction \* 2;

// .1..2..3..4.

double Y = Y\_s + (Y\_e - Y\_s) / 8;

double Y2 = Y\_s + (Y\_e - Y\_s) / 8 \* 3;

double Y3 = Y\_s + (Y\_e - Y\_s) / 8 \* 5;

double Y4 = Y\_s + (Y\_e - Y\_s) / 8 \* 7;

if (Y2 - Y < min) { Y = Y\_m - min / 2 \* 3; Y2 = Y\_m - min / 2; Y3 = Y\_m + min / 2; Y4 = Y\_m + min / 2 \* 3; }

double PinWidth = base\_size - EllipseSize + PinStrokeSize;

return new Point[][] {

new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход

new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход

new Point[] { new(X, Y3), new(X + PinWidth, Y3) }, // Третий вход

new Point[] { new(X, Y4), new(X + PinWidth, Y4) }, // Четвёртый вход

new Point[] { new(X2, Y2), new(X2 + PinWidth, Y2) }, // Первый выход

new Point[] { new(X2, Y3), new(X2 + PinWidth, Y3) }, // Второй выход

};

} }

/\*

\* Мозги

\*/

public void Brain(ref bool[] ins, ref bool[] outs) {

bool a = ins[0], b = ins[1], c = ins[2], d = ins[3];

if (d) {

outs[0] = true;

outs[1] = true;

} else if (c) {

outs[0] = false;

outs[1] = true;

} else if (b) {

outs[0] = true;

outs[1] = false;

} else if (a) {

outs[0] = false;

outs[1] = false;

}

}

}

}