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 316

Выполнил:

Студент группы ИП-117

Третьяков М.О.

Проверил:

Ассистент кафедры ПМ и К

Меркулов И. А.

Новосибирск 2023

Оглавление

Техническое задание

Реализовать приложение-симулятор логических схем.

Работа состоит из следующих этапов:

1. Создание Use-Case диаграммы приложения. По окончании этапа должны быть построены UseCase диаграммы.

2. Разработка графического интерфейса (схематичное изображение интерфейса и описание возможностей элементов, достижения сценариев описанных в Use-Case диаграмме посредством этих элементов). По окончании этапа должна быть построена схема интерфейса с подробным описанием элементов и достижения сценариев из use-case диаграммы.

3. Проектирование приложения - создание ER-диаграмм, диаграмм классов. По окончании этапа должны быть построены диаграммы классов с описанием (обязательно), ER-диаграммы (необязательно).

4. Разработка. При разработке используется TDD и упрощённый git flow (одна функциональность - одна ветка, коммиты в логических точках).

В репозитории приложения должен находиться отчёт по первым трём пунктам и проекты с исходным кодом и юнит-тестами.

Реализация

В ходе выполнения поставленного технического задания, мною были реализованы следующие элементы программы:

Главное меню

При запуске программы пользователь видит главное меню, в котором присутствуют несколько кнопок:

1. Создать новый проект

   При активации пользователю открывается основное окно программы, в которой непосредственно происходит построение логических схем

2. Открыть проект

   При активации открывается Windows интерфейс выбора файла, после чего пользователя переносит в основное окно программы

3. Выход

   При активации программа закрывается

Так же под кнопками есть список уже ранее открытых/созданных схем. Если же ранее схемы не редактировались, то список будет пуст.

Основное окно программы

После открытия или создания схемы пользователь попадает в окно редактора. Вверху находятся кнопки меню, при активации которых выпадает контекстное меню:

1. Файл
   a. Создать новую схему
   b. Открыть проект
   c. Сохранить текущую схему
   d. Сохранить проект как
   e. Выйти в главное меню
   f. Выйти
2. Опции
   a. Блокировать самоподключение

Слева находится панель логических элементов, справа - дерево проекта, со списком схем, при желании можно создать новую схему, удалить уже имеющуюся или перенести схему на уровень выше. Посередине находится холст для неподстредсвенного построения схемы.

Чтобы добавить новый элемент на холст необходимо выбрать его в панели элементов слева, после чего нажать на холст в нужном месте. Чтобы соединить элементы требуется выбрать нужный выход и подключить его ко входу другого элемента, выход к выходу или вход ко входу подключить нельзя. Для удаления элемента с холста нужно выбрать его левой кнопкой мыши, после чего нажать клавишу “delete” на клавиатуре. С помощью колесика мыши можно увеличить выбранный элемент или изменить размер холста.

Пример работы программы

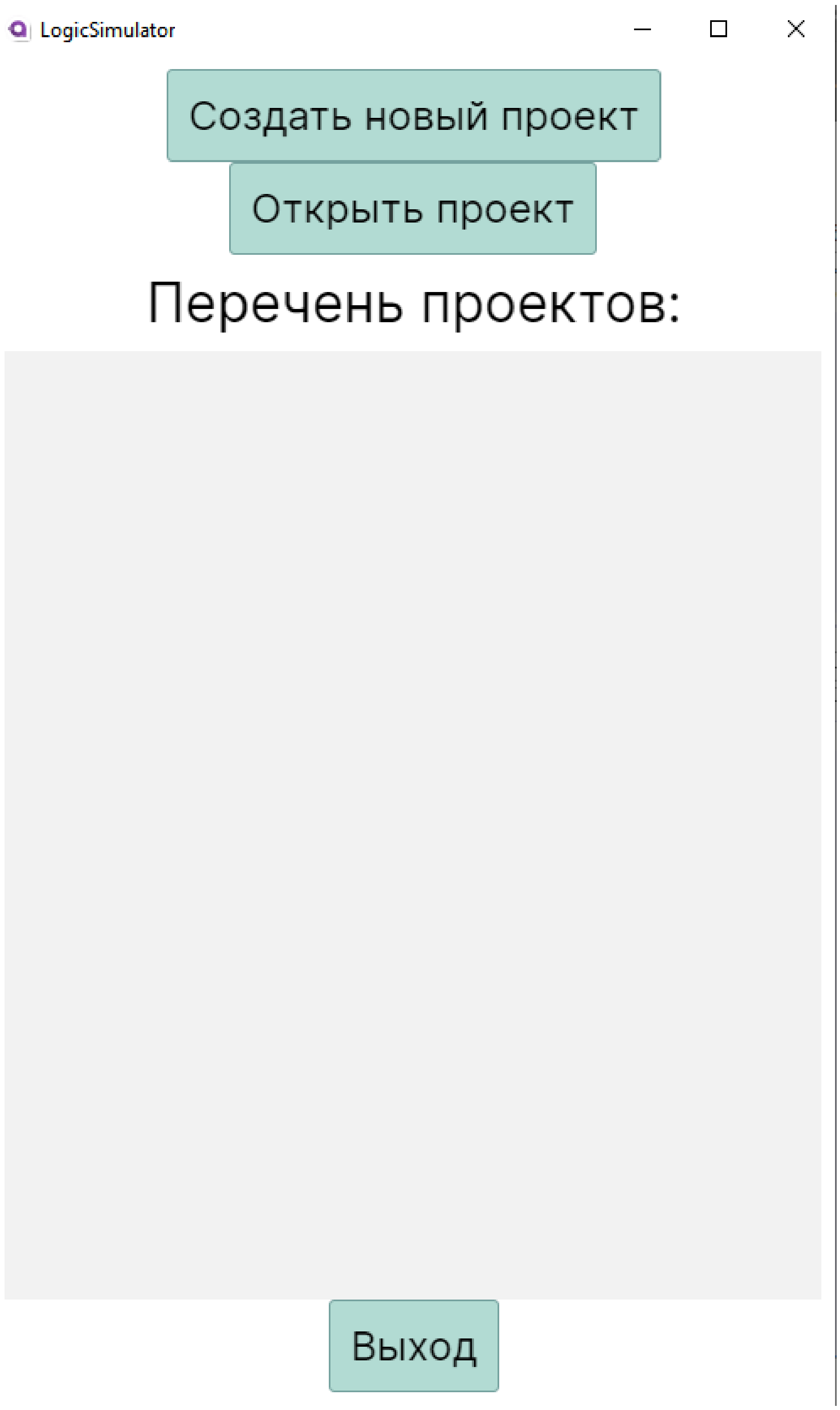

Рис. 1 - Главное меню программы

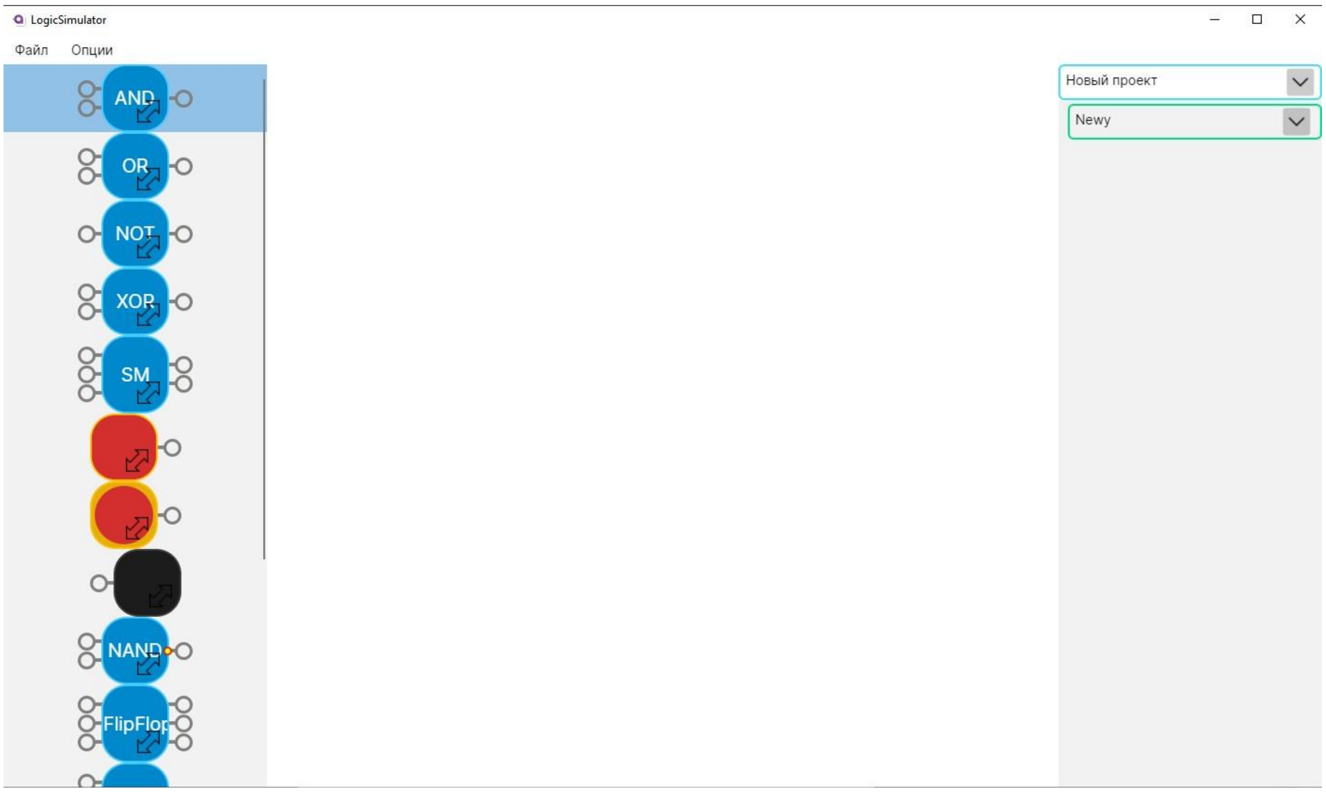

Рис. 2 - Главное окно программы

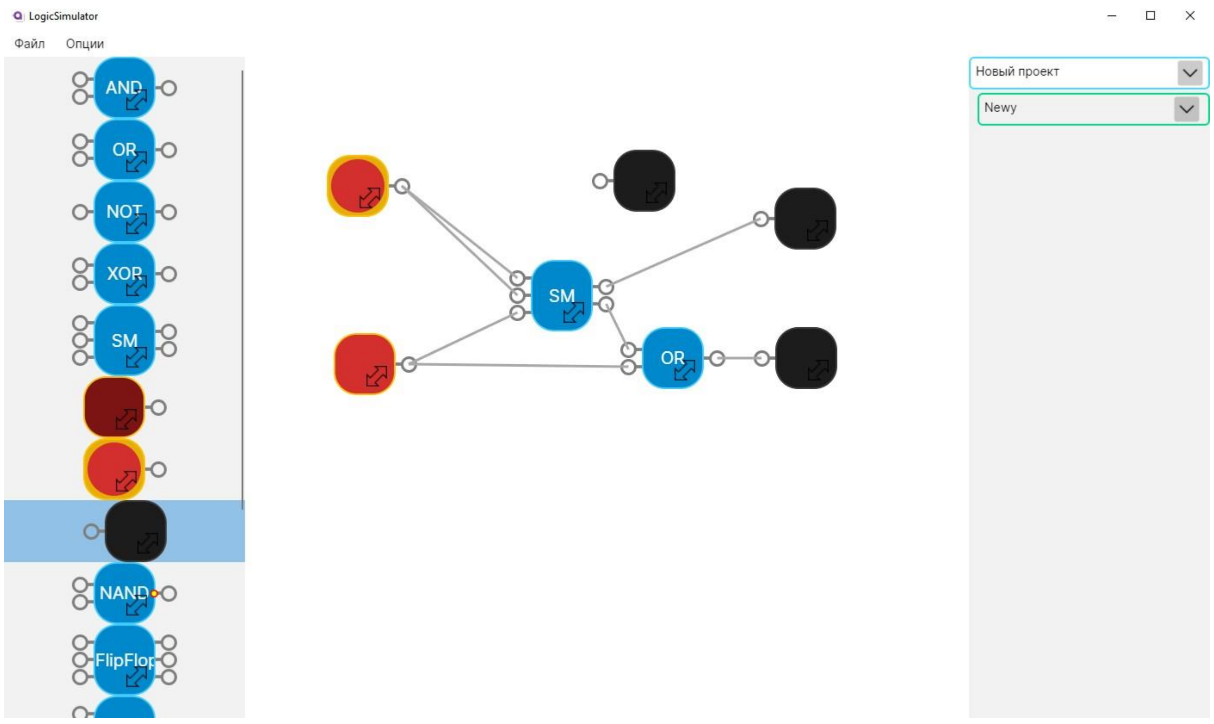

Рис. 3 - Пример построенной схемы

Диаграммы и схемы

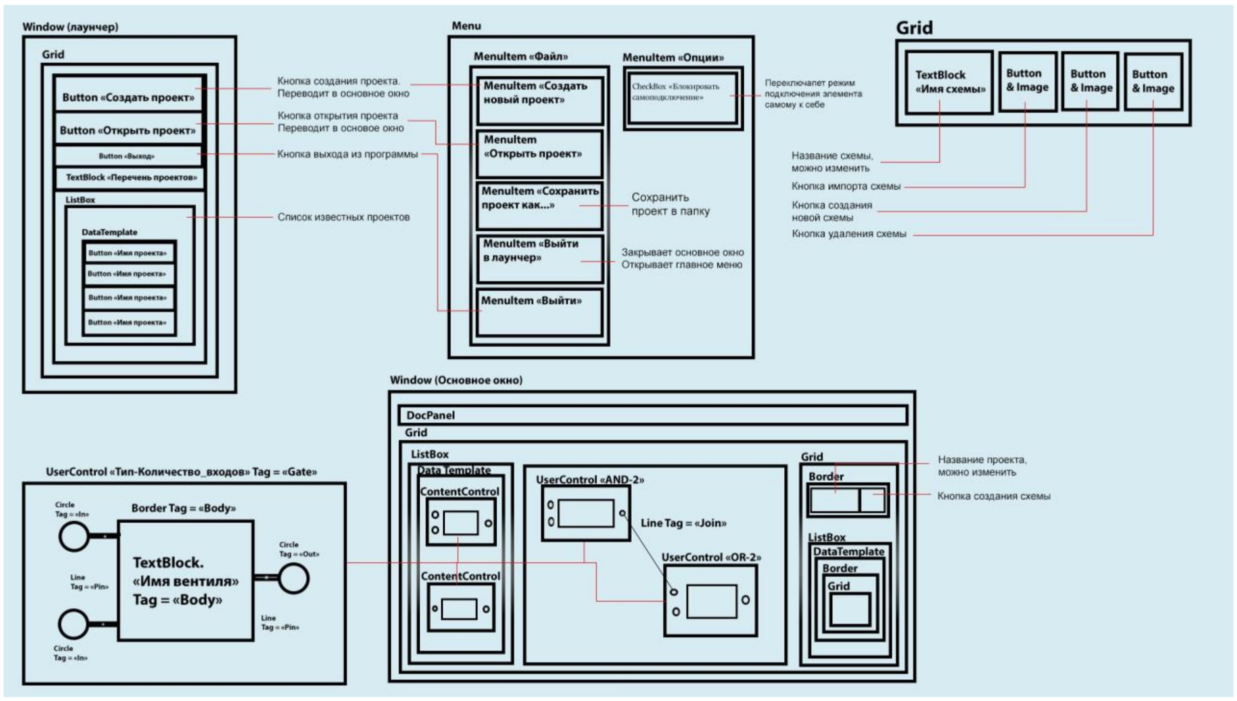
Window (лаунчер)
Grid
Button «Создать проект»
Button «Открыть проект»
Button «Выход»
TextBlock «Перечень проектов»
ListBox
DataTemplate
Button «Имя проекта»
Кнопка создания проекта. Переводит в основное окно
Кнопка открытия проекта Переводит в основное окно
Кнопка выхода из программы
Список известных проектов
Menu
MenuItem «Файл»
MenuItem «Создать новый проект»
MenuItem «Открыть проект»
MenuItem «Сохранить проект как...»
MenuItem «Выйти в лаунчер»
MenuItem «Выйти»
MenuItem «Опции»
CheckBox «Блокировать самоподключение»
Переключает режим подключения элемента самому к себе
Сохранить проект в папку
Закрывает основное окно Открывает главное меню
Grid
TextBlock «Имя схемы»
Button & Image
Название схемы, можно изменить
Кнопка импорта схемы
Кнопка создания новой схемы
Кнопка удаления схемы
Window (Основное окно)
DocPanel
Grid
ListBox
Data Template
ContentControl
UserControl «AND-2»
Line Tag = «Join»
UserControl «OR-2»
Border
DataTemplate
Название проекта, можно изменить
Кнопка создания схемы
UserControl «Тип-Количество_входов» Tag = «Gate»
Circle Tag = «In»
Border Tag = «Body»
TextBlock. «Имя вентиля» Tag = «Body»
Circle Tag = «Out»
Line Tag = «Pin»

Рис. 1 - схема интерфейса

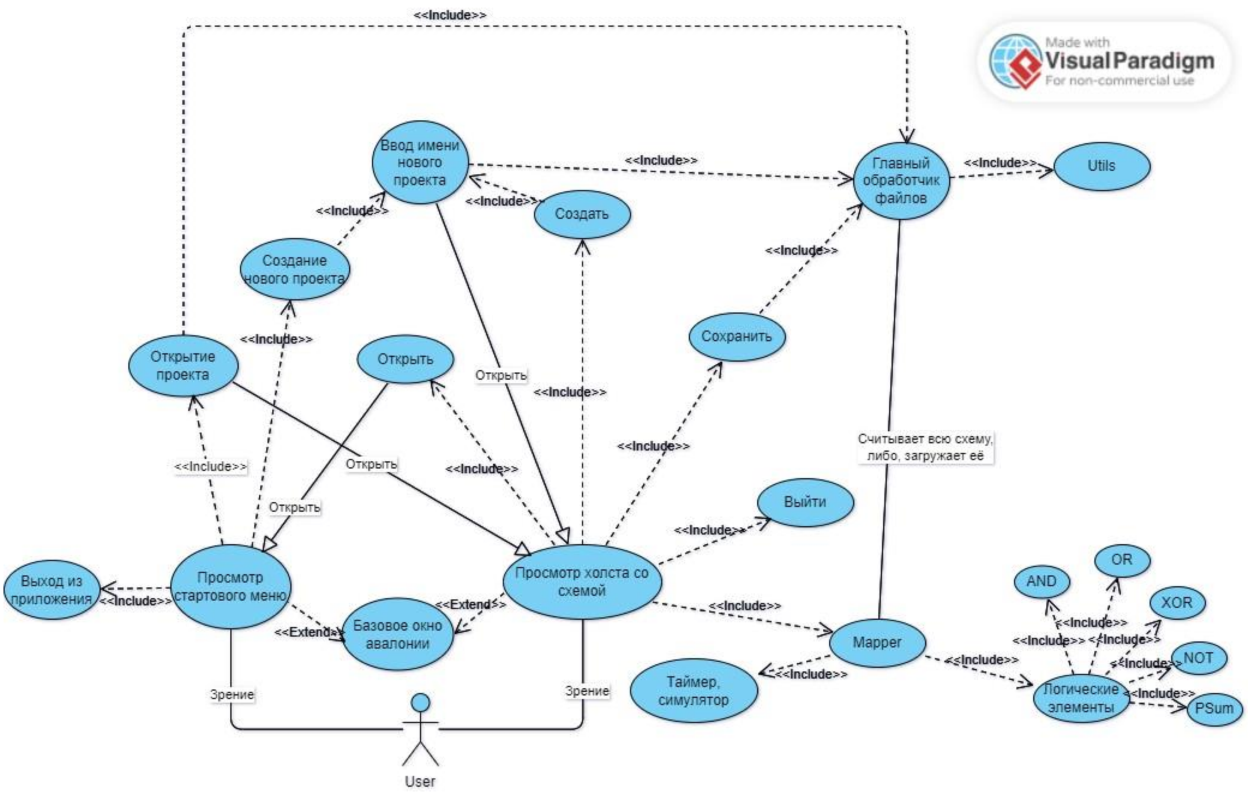

Рис. 2 - Use-Case диаграмма

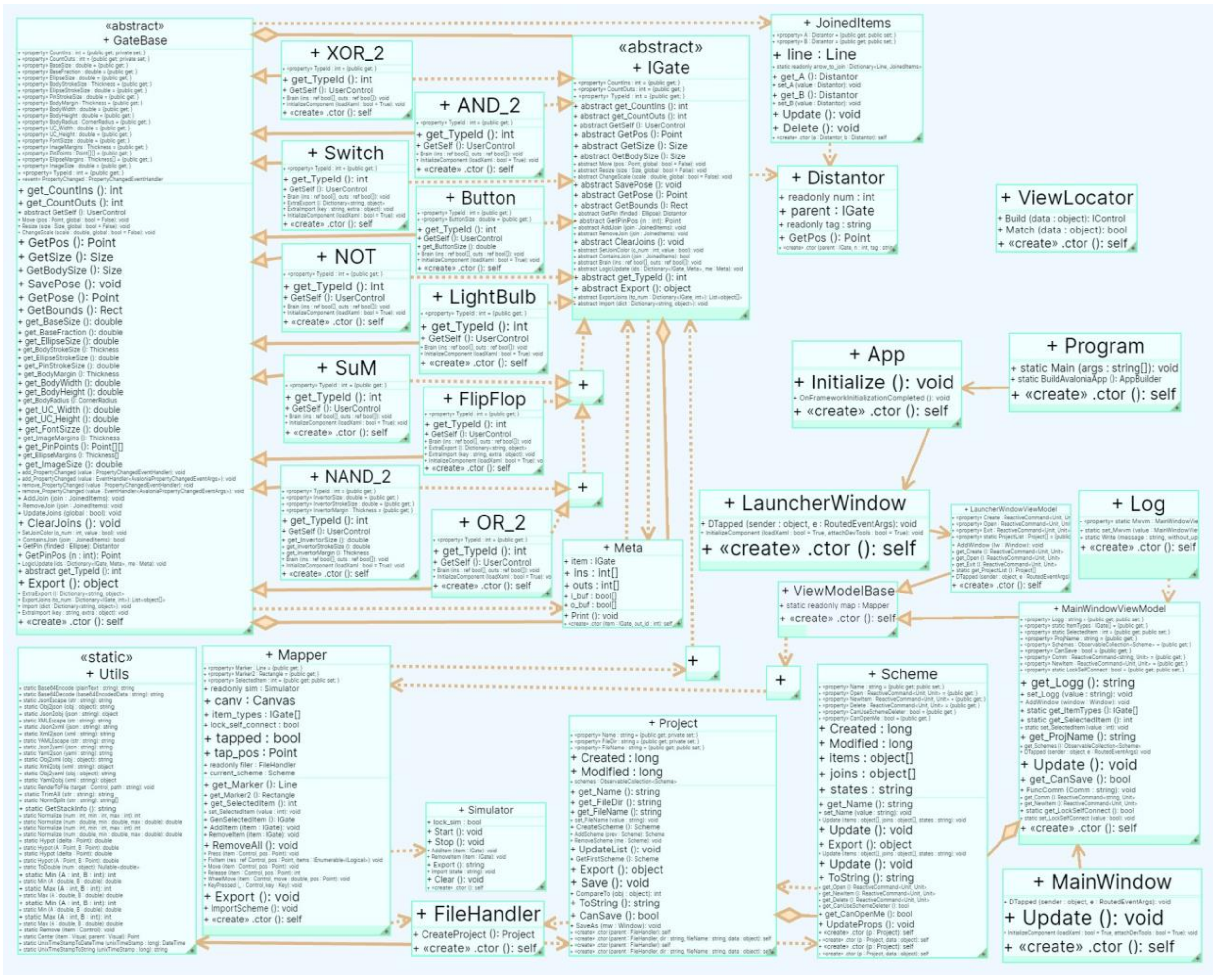

Рис. 3 - ER-диаграмма

Вывод

В рамках проектной работы были усовершенствованы навыки, полученные в курсе "Визуальное программирование и человеко-машинное взаимодействие", такие как разработка пользовательских интерфейсов, модульное тестирование графических интерфейсов, создание шаблонов отображения и многостраничных приложений.

Также были улучшены навыки работы с графической трансформацией и анимацией, обработкой событий с устройств ввода и созданием собственных элементов управления. Были отработаны навыки использования GUI Framework Avalonia UI на платформе .NET с помощью языка программирования C#. Улучшено понимание принципов объектно ориентированного подхода к написанию программ. Таким образом, были достигнуты значительные успехи в работе над проектом.

Исходный код

Файл Distantor.cs

```
using Avalonia;
using LogicSimulator.Views.Shapes;

namespace LogicSimulator.Models
{
    public class Distantor
    {
        public readonly int num;
        public IGate parent;
        public readonly string tag;

        public Distantor(IGate parent, int n, string tag)
        {
            this.parent = parent;
            num = n;
            this.tag = tag;
        }

        public Point GetPos() => parent.GetPinPos(num);
    }
}
```

Файл FileHandler.cs

```
using Avalonia.Controls;
using LogicSimulator.ViewModels;
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;
using System.Linq;

namespace LogicSimulator.Models
{
    public class FileHandler
    {
        readonly string AppData;
        readonly List<Project> projects = new();
        readonly List<string> project_paths = new();
        readonly Dictionary<string, Project> proj_dict = new();

        public FileHandler()
        {

            string app_data =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
            app_data = Path.Combine(app_data, "LogicSimulator");
            if (!Directory.Exists(app_data))
                Directory.CreateDirectory(app_data);
            AppData = app_data;
            LoadProjectList();
        }

        private void AddProject(Project proj)
        {
            if (proj.FileDir == null || proj.FileName == null)
                return;

            var path = Path.Combine(proj.FileDir, proj.FileName);
            if (proj_dict.ContainsKey(path))
                return;

            proj_dict[path] = proj;
            projects.Add(proj);
        }
        private static string GetProjectFileName(string dir)
        {
            int n = 0;
            while (true) {
                string name = "proj_" + ++n + ".json";
                if (!File.Exists(Path.Combine(dir, name)))
                    return name;
            }
        }

        public Project CreateProject()
        {
            var proj = new Project(this);
            return proj;
        }
        private Project? LoadProject(string dir, string fileName)
        {
            try
            {
                var path = Path.Combine(dir, fileName);
                if (!File.Exists(path))
                    return null;


                var obj = Utils.Json2obj(File.ReadAllText(path)) ?? throw new
DataException("Invalid project JSON file structure!");
                var proj = new Project(this, dir, fileName, obj);
                AddProject(proj);
                return proj;
            }
            catch (Exception e)
            {
                Log.Write("Unsuccessful attempt to load project:\n" + e);
            }
            return null;
        }
        private Project? LoadProject(string path)
        {
            var s_arr = path.Split(Path.DirectorySeparatorChar).ToList();
```

```
                var name = s_arr[^1];
                s_arr.RemoveRange(s_arr.Count - 1, 1);
                var dir = Path.Combine(s_arr.ToArray());

                return LoadProject(dir, name);
            }
            private void LoadProjectList()
            {
                var file = Path.Combine(AppData, "project_list.db");
                if (!File.Exists(file))
                    return;

                string[] data;
                try {

                    data = Utils.SQLite_proj_list2obj(file) ?? throw new
        DataException("Incorrect structure of the SQLite (.db) project list file!");
                }
                catch (Exception e)
                {
                    Log.Write("Failed to load list of project:\n" + e);
                    return;
                }
                foreach (var path in data)
                {
                    project_paths.Add(path);
                    LoadProject(path);
                }
            }

            internal static void SaveProject(Project proj)
            {
                var dir = proj.FileDir;
                if (dir == null)
                    return;

                var data = Utils.Obj2json(proj.Export());
                var name = proj.FileName;
                name ??= GetProjectFileName(dir);
                proj.FileName = name;

                var path = Path.Combine(dir, name);
                File.WriteAllText(path, data);
            }
            private void SaveProjectList()
            {
                var file = Path.Combine(AppData, "project_list.db");
                if (Path.Exists(file))
                    File.WriteAllBytes(file, Array.Empty<byte>());
                Utils.Obj2sqlite_proj_list(project_paths.ToArray(), file);
            }

            internal Project[] GetSortedProjects()
            {
                projects.Sort();
                return projects.ToArray();
```

```
}
```

```
        internal void AppendProject(Project proj)
        {
            if (proj.FileDir == null || proj.FileName == null)
                return;

            var path = Path.Combine(proj.FileDir, proj.FileName);
            if (project_paths.Contains(path))
                return;

            project_paths.Add(path);
            AddProject(proj);
            SaveProjectList();
        }

        internal static string? RequestProjectPath(Window parent)
        {
            var dlg = new OpenFolderDialog
            {
                Title = "Select folder"
            };
            var task = dlg.ShowAsync(parent);
            return task.GetAwaiter().GetResult();
        }
        internal Project? SelectProjectFile(Window parent)
        {
            var dlg = new OpenFileDialog
            {
                Title = "Select json file"
            };

            dlg.Filters?.Add(new FileDialogFilter() { Name = "JSON Files",
Extensions = { "json" } });

            dlg.Filters?.Add(new FileDialogFilter() { Name = "All Files",
Extensions = { "*" } });
            dlg.AllowMultiple = false;

            var task = dlg.ShowAsync(parent);
            var res = task.GetAwaiter().GetResult();
            if (res == null)
                return null;

            var path = res[0];
            if (!proj_dict.TryGetValue(path, out var proj))
            {
                proj = LoadProject(path);
                if (proj != null) AppendProject(proj);
            }
            return proj;
        }
    }
}
```

Файл JoinedItems.cs

```
using Avalonia.Controls.Shapes;
using Avalonia.Media;
using System.Collections.Generic;

namespace LogicSimulator.Models {
    public class JoinedItems
    {
new();  public static readonly Dictionary<Line, JoinedItems> arrow_to_join =

        public JoinedItems(Distantor a, Distantor b)
        {
            A = a;
            B = b;
            Update();
            a.parent.AddJoin(this);
            if (a.parent != b.parent)
                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);
        }
    }
}
```

Файл 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 Avalonia.Input;

namespace LogicSimulator.Models
{
    public class Mapper
    {

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

        readonly Rectangle marker2 = new() { Tag = "Marker", Classes =
new("anim"), ZIndex = 2, IsVisible = false, Stroke =
Brushes.MediumAquamarine, StrokeThickness = 3 };

        public Line Marker { get => marker; }
        public Rectangle Marker2 { get => marker2; }

        public readonly Simulator sim = new();

        public Canvas canv = new();

        private IGate? marked_item;
        private JoinedItems? marked_line;

        private void UpdateMarker()
        {
            marker2.IsVisible = marked_item != null || marked_line != null;

            if (marked_item != null)
            {
                var bound = marked_item.GetBounds();
                marker2.Margin = new(bound.X, bound.Y);
                marker2.Width = bound.Width;
                marker2.Height = bound.Height;
                marked_line = null;
            }

            if (marked_line != null)
            {
                var line = marked_line.line;
                var A = line.StartPoint;
                var B = line.EndPoint;
                marker2.Margin = new(Math.Min(A.X, B.X), Math.Min(A.Y, B.Y));
                marker2.Width = Math.Abs(A.X - B.X);
                marker2.Height = Math.Abs(A.Y - B.Y);
            }
        }

        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 SuM(),
                5 => new Switch(),
                6 => new Button(),
                7 => new LightBulb(),
                8 => new NAND_2(),
                9 => new FlipFlop(),
                10 => new OR_8(),
                11 => new AND_8(),
                _ => new AND_2(),
            };
        }


        public IGate[] item_types = Enumerable.Range(0,
12).Select(CreateItem).ToArray();

        public IGate GenSelectedItem() => CreateItem(selected_item);

        readonly List<IGate> items = new();

        private void AddToMap(IControl item)
        {
            canv.Children.Add(item);
        }

        public void AddItem(IGate item)
        {
            items.Add(item);
            sim.AddItem(item);
            AddToMap(item.GetSelf());
        }
        public void RemoveItem(IGate item)
        {
            if (marked_item != null)
            {
                marked_item = null;
                UpdateMarker();
            }
            if (marked_line != null && item.ContainsJoin(marked_line))
            {
                marked_line = null;
                UpdateMarker();
            }

            items.Remove(item);
            sim.RemoveItem(item);

            item.ClearJoins();
            ((Control) item).Remove();
        }
        public void RemoveAll()
        {
```

```
foreach (var item in items.ToArray())
```

```
                RemoveItem(item);
            sim.Clear();
        }

        private void SaveAllPoses()
        {
            foreach (var item in items)
                item.SavePose();
        }

        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;
        }

        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 bool lock_self_connect = true;

        public void Press(Control item, Point pos)
        {
            UpdateMode(item);

            moved_pos = pos;
            moved_item = GetGate(item);
            tapped = true;
            if (moved_item != null)
                item_old_pos = moved_item.GetPos();

            switch (mode)
            {
                case 1:
                    SaveAllPoses();
                    break;
                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("Unexpected error"); // Такого не бывает
                    start_dist = gate.GetPin(marker_circle);

                    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;
@join2);            JoinedItems.arrow_to_join.TryGetValue(@join, out var
                    if (@join2 == null) break;

                    if (marked_line == @join2) {
                        marked_line = null;
                        UpdateMarker();
                    }

                    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;
@join2.B.tag);      marker_mode = CalcMode(join_start ? @join2.A.tag :

                    marker.IsVisible = true;
                    @join.IsVisible = false;
                    break;
                }

                Move(item, pos);
            }

items)      public void FixItem(ref Control res, Point pos, IEnumerable<ILogical>
            {
                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, bool use_fix = true)
            {
8))             if (use_fix && (mode == 5 || mode == 6 || mode == 7 || mode ==
                {
                    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;
```

```
            if (IsMode(item, mods) && item is Ellipse @ellipse
tag == "Out" || && !(marker_mode == 5 && tag == "In" || marker_mode == 6 &&
                lock_self_connect && moved_item == GetGate(item)))
            {

                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;

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

            switch (mode)
            {
                case 1:
                    foreach (var item_ in items)
                    {
                        var pose = item_.GetPose();
                        item_.Move(pose + delta, true);
                    }
                    UpdateMarker();
                    break;
                case 2:
                    if (moved_item == null)
                        break;
                    var new_pos = item_old_pos + delta;
                    moved_item.Move(new_pos);
                    UpdateMarker();
                    break;
                case 3:
                    if (moved_item == null)
                        break;
delta.Y);           var new_size = item_old_size + new Size(delta.X,
                    moved_item.Resize(new_size);
```

```
Updat
eMark
er();
```

```
                    break;
                case 5 or 6 or 7:
                    var end_pos = marker_circle == null ? pos :
marker_circle.Center(canv);
                    marker.EndPoint = end_pos;
                    break;
                case 8:
                    if (old_join == null)
                        break;

                    var p = marker_circle == null ? pos :
marker_circle.Center(canv);
                    if (join_start)
                        marker.EndPoint = p;
                    else
                        marker.StartPoint = p;
                    break;
            }
        }

        public bool tapped = false;
        public Point tap_pos;

        public int Release(Control item, Point pos, bool use_fix = true)
        {
            Move(item, pos, use_fix);

            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("Unexpected error");
                    var end_dist = gate.GetPin(marker_circle);
                    var newy = new JoinedItems(start_dist, end_dist);
                    AddToMap(newy.line);
                }
                marker.IsVisible = false;
                marker_mode = 0;
                break;
            case 8:
                if (old_join == null)
                    break;
@join);         JoinedItems.arrow_to_join.TryGetValue(old_join, out var
                if (marker_circle != null && @join != null)
                {

                    var gate = GetGate(marker_circle) ?? throw new
Exception("Unexpected error");
                    var p = gate.GetPin(marker_circle);
                    @join.Delete();
                    var newy = join_start ? new JoinedItems(@join.A, p) : new
JoinedItems(p, @join.B);
                    AddToMap(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;
            moved_item = null;
            return res_mode;
        }

        private void Tapped(Control item, Point pos)
        {
            tap_pos = pos;

            switch (mode)
            {
                case 2 or 8:
                    if (item is Line @line)
                    {
                        if (!JoinedItems.arrow_to_join.TryGetValue(@line, out
var @join))
                            break;
                        marked_item = null;
                        marked_line = @join;
                        UpdateMarker();
                        break;
                    }

                    if (moved_item == null)
                        break;

                    marked_item = moved_item;
                    UpdateMarker();
                    break;
            }
        }

        public void WheelMove(Control item, double move, Point pos)
        {
            int mode = CalcMode((string?) item.Tag);
```

```
double scale = move > 0 ? 1.1 : 1 / 1.1;
```

```
            double inv_scale = 1 / scale;

            switch (mode)
            {
                case 1:
                    foreach (var gate in items) {
                        gate.ChangeScale(scale, true);

                        var item_pos = gate.GetPos();
                        var delta = item_pos - pos;
                        delta *= scale;
                        var new_pos = delta + pos;
                        gate.Move(new_pos, false);
                    }
                    UpdateMarker();
                    break;
                case 2:
                    var gate2 = GetGate(item);
                    if (gate2 == null)
                        return;
                    gate2.ChangeScale(inv_scale);
                    UpdateMarker();
                    break;
            }
        }

        public void KeyPressed(Control _, Key key)
        {
            switch (key) {
                case Key.Up:
                case Key.Left:
                case Key.Right:
                case Key.Down:
                    int dx = key == Key.Left ? -1 : key == Key.Right ? 1 : 0;
                    int dy = key == Key.Up ? -1 : key == Key.Down ? 1 : 0;
                    marked_item?.Move(marked_item.GetPos() + new Point(dx *
10, dy * 10));
                    UpdateMarker();
                    break;
                case Key.Delete:
                    if (marked_item != null) RemoveItem(marked_item);
                    if (marked_line != null)
                    {
                        marked_line.Delete();
                        marked_line = null;
                        UpdateMarker();
                    }
                    break;
            }
        }

        public readonly FileHandler filer = new();
        public Scheme? current_scheme;

        public void Export()
        {
            if (current_scheme == null)
```

```
return;
```

```
    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));

    sim.Clean();
    string states = sim.Export();

    try
    {
        current_scheme.Update(arr, joins.ToArray(), states);
    }
    catch (Exception e)
    {
        Log.Write("Save error:\n" + e);
    }
}

public void ImportScheme(bool start = true)
{
    if (current_scheme == null)
        return;

    sim.Stop();
    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("Invalid element type: " + item);
            continue;
        }

        if (!@dict.TryGetValue("id", out var @value))
        {
            Log.Write("Element id not found");
            continue;
        }
        if (@value is not int @id)
        {
            Log.Write("Invalid id type: " + @value);
            continue;
        }
```

```
                var newy = CreateItem(@id);

                newy.Import(@dict);
                AddItem(newy);
                list.Add(newy);
            }
            var items_arr = list.ToArray();

            List<JoinedItems> joinz = new();
            foreach (var obj in current_scheme.joins)
            {
                object[] join;
                if (obj is List<object> @j)
                {
                    join = @j.ToArray();
                }
                else if (obj is object[] @j2)
                {
                    join = @j2;
                }
                else
                {
                    Log.Write("One of the wrong connections: " + obj + " " +
Utils.Obj2json(obj));
                    continue;
                }
                if (join.Length != 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("Connection list content is wrong");
                    continue;
                }


                var newy = new JoinedItems(new(items_arr[@num_a], @pin_a,
tag_a), new(items_arr[@num_b], @pin_b, tag_b));
                AddToMap(newy.line);
                joinz.Add(newy);
            }

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

            sim.Import(current_scheme.states);
            sim.lock_sim = false;
            if (start)
                sim.Start();
        }
    }
}
```

Файл Project.cs

```
using Avalonia.Controls;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;

namespace LogicSimulator.Models
{
    public class Project: IComparable
    {
        public string Name { get; private set; }
        public long Created;
        public long Modified;

        public ObservableCollection<Scheme> schemes = new();
        public string? FileDir { get; private set; }
        public string? FileName { get; set; }

        private readonly FileHandler parent;

        public Project(FileHandler parent)
        {
            this.parent = parent;
            Name = "Новый проект";
            Created = Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
            FileDir = null;
            FileName = null;
            CreateScheme();
        }


        public Project(FileHandler parent, string dir, string fileName,
object data)
        {
            this.parent = parent;
            FileDir = dir;
            FileName = fileName;

            if (data is not Dictionary<string, object> dict)
                throw new Exception("Expected dictionary at project root");

            if (!dict.TryGetValue("name", out var value))
                throw new Exception("The project has no name");
            if (value is not string name)
                throw new Exception("Project name type is not a string");

            Name = name;

            if (!dict.TryGetValue("created", out var value2))
                throw new Exception("The project has no creation time");
            if (value2 is not int create_t)
                throw new Exception("Project creation time is not a string");
            Created = create_t;

            if (!dict.TryGetValue("modified", out var value3))
project");      throw new Exception("There is no change time in the

            if (value3 is not int mod_t)
string");       throw new Exception("Project modification time is not a

            Modified = mod_t;

            if (!dict.TryGetValue("schemes", out var value4))
list");         throw new Exception("The project does not have a schema
            if (value4 is not List<object> arr)
of strings");   throw new Exception("List of project schemas - not an array

            foreach (var s_data in arr)
            {

                if (s_data == null) throw new Exception("One of the file
names of the list of project schemas is null");
                var scheme = new Scheme(this, s_data);
                schemes.Add(scheme);
            }
        }



        public Scheme CreateScheme()
        {
            var scheme = new Scheme(this);
            schemes.Add(scheme);
            Save();
            return scheme;
        }
        public Scheme AddScheme(Scheme? prev)
        {
            var scheme = new Scheme(this);
            int pos = prev == null ? 0 : schemes.IndexOf(prev) + 1;
            schemes.Insert(pos, scheme);
            Save();
            return scheme;
        }
        public void RemoveScheme(Scheme me)
        {
            schemes.Remove(me);
            Save();
        }
        public void UpdateList()
        {
            foreach (var scheme in schemes)
                scheme.UpdateProps();
        }

        public Scheme GetFirstScheme() => schemes[0];
        public object Export()
        {
            return new Dictionary<string, object>
            {
                ["name"] = Name,
                ["created"] = Created,
                ["modified"] = Modified,
                ["schemes"] = schemes.Select(x => x.Export()).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 + "\nChanged: " + Modified.UnixTimeStampToString() +
"\nCreated: " + Created.UnixTimeStampToString();
        }

        internal void ChangeName(string name)
        {
            Name = name;
            Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
            Save();
        }

        public bool CanSave() => FileDir != null;
        public void SaveAs(Window mw)
        {
            FileDir = FileHandler.RequestProjectPath(mw);
            Save();
            parent.AppendProject(this);
        }

        public void SetDir(string path) => FileDir = path;
    }
}
```

Файл Scheme.cs

```
using LogicSimulator.ViewModels;
using ReactiveUI;
using System;
using System.Collections.Generic;
using System.Reactive;

namespace LogicSimulator.Models
{
    public class Scheme : ReactiveObject
    {
        public string Name { get; set; }
        public long Created;
        public long Modified;

        public object[] items;
        public object[] joins;
        public string states;

        private readonly Project parent;

        public Scheme(Project p)
        {
            Created = Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
            Name = "New";
            items = joins = Array.Empty<object>();
            states = "0";
            parent = p;

            Open = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncOpen();
                return new Unit();
            });
            NewItem = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncNewItem();
                return new Unit();
            });
            Delete = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncDelete();
                return new Unit();
            });
        }

        public Scheme(Project p, object data)
        {
            parent = p;

            if (data is not Dictionary<string, object> dict)
                throw new Exception("Expected dictionary at schema root");

            if (!dict.TryGetValue("name", out var value))
                throw new Exception("There is no name in the schema");
            if (value is not string name)
                throw new Exception("Schema name type is not a string");

            Name = name;

            if (!dict.TryGetValue("created", out var value2))
schema");       throw new Exception("There is no creation time in the
```

```
            if (value2 is not int create_t)
                throw new Exception("Schema creation time is not a string");

            Created = create_t;

            if (!dict.TryGetValue("modified", out var value3))
                throw new Exception("There is no change time in the schema");
            if (value3 is not int mod_t)
                throw new Exception("Schema change time is not a string");

            Modified = mod_t;

            if (!dict.TryGetValue("items", out var value4))
elements");     throw new Exception("The schema does not have a list of
            if (value4 is not List<object> arr)
of objects");   throw new Exception("List of schema elements - not an array

            items = arr.ToArray();

            if (!dict.TryGetValue("joins", out var value5))
                throw new Exception("There is no netlist in the diagram");
            if (value5 is not List<object> arr2)
objects");      throw new Exception("Schematic netlist - not an array of

            joins = arr2.ToArray();

            if (!dict.TryGetValue("states", out var value6))
schema");       throw new Exception("There is no list of states in the
            if (value6 is not string arr3)
                throw new Exception("List of schema states - not a string");

            states = arr3;

            Open = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncOpen();
                return new Unit();
            });
            NewItem = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncNewItem();
                return new Unit();
            });
            Delete = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncDelete();
                return new Unit();
            });
        }

        public void Update(object[] items, object[] joins, string 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 Update()
        {
            Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
            parent.Modified = Modified;
            parent.Save();
        }

        public override string ToString() => Name;

        internal void ChangeName(string name)
        {
            Name = name;
            Update();
        }

        void FuncOpen()
        {
            ViewModelBase.map.current_scheme = this;
            ViewModelBase.map.ImportScheme();
            parent.UpdateList();
        }
        void FuncNewItem()
        {
            parent.AddScheme(this);
            parent.UpdateList();
        }
        void FuncDelete() {
            parent.RemoveScheme(this);
            parent.UpdateList();
        }

        public ReactiveCommand<Unit, Unit> Open { get; }
        public ReactiveCommand<Unit, Unit> NewItem { get; }
        public ReactiveCommand<Unit, Unit> Delete { get; }

        public bool CanUseSchemeDeleter { get => parent.schemes.Count > 1; }
this; } public bool CanOpenMe { get => ViewModelBase.map.current_scheme !=

        public void UpdateProps()
```

{

```
                this.RaisePropertyChanged(nameof(CanUseSchemeDeleter));
                this.RaisePropertyChanged(nameof(CanOpenMe));
            }
        }
    }
```

Файл Simulator.cs

```
using LogicSimulator.ViewModels;
using LogicSimulator.Views.Shapes;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
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("Element: " + item + " | Ins: " + Utils.Obj2json(ins) +
" | Outs: " + Utils.Obj2json(outs));
        }
    }


    public class Simulator
    {
        public Simulator()
        {
            Start();
        }

        private Task? task;
        private bool stop_sim = false;
```

```
public bool lock_sim = false;
public void Start()
{
    if (task != null || lock_sim)
        return;
    stop_sim = false;
    task = Task.Run(async () =>
    {
        for (; ; ) {
            await Task.Delay(1);

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

            if (stop_sim) return;
        }
    });
}
public void Stop()
{
    if (task == null)
        return;
    stop_sim = true;
    task.GetAwaiter().GetResult();
    task = null;
}



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)
{
    Stop();

    int out_id = outs.Count;
    for (int i = 0; i < item.CountOuts; i++)
    {
        outs.Add(false);
        outs2.Add(false);
    }

    Meta meta = new(item, out_id);
    items.Add(meta);
    ids.Add(item, meta);
```

```
Start();
```

```
        }

        public void RemoveItem(IGate item)
        {
            Stop();

            Meta meta = ids[item];
            meta.item = null;
            foreach (var i in Enumerable.Range(0, meta.outs.Length))
            {
                int n = meta.outs[i];
                outs[n] = outs2[n] = false;
            }
            ids.Remove(item);

            Start();
        }

        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);

            if (comparative_test_mode)
            {
                prev_state = cur_state;
                cur_state = Export();
            }
        }

        public void Clean()
        {
            int n = 0;
```

```
int[] arr = Enumerable.Repeat(-1, outs.Count).ToArray();
```

```
            StringBuilder sb = new();
            sb.Append(’0’);
            foreach (var meta in items)
                if (meta.item != null)
                    foreach (var @out in meta.outs)
                    {
                        arr[@out] = ++n;
                        sb.Append(outs[@out] ? ’1’ : ’0’);
                    }
            arr[0] = 0;
            foreach (var meta in items)
            {
                meta.outs = meta.outs.Select(x => arr[x]).ToArray();
                meta.ins = meta.ins.Select(x => arr[x]).ToArray();
            }
            Import(sb.ToString());
        }
        public string Export() => string.Join(″″, outs.Select(x => x ? ’1’ :
’0’));
        public void Import(string state)
        {
            if (state.Length == 0)
                state = ″0″;
            outs = state.Select(x => x == ’1’).ToList();
            outs2 = outs.ToList(); // clone
        }
        public void Clear()
        {
            outs = new() { false };
            outs2 = new() { false };
            items.Clear();
            ids.Clear();
        }

        public void TopSecretPublicTickMethod() => Tick();

        public Switch[] GetSwitches() => items.Select(x =>
x.item).OfType<Switch>().ToArray();

        public LightBulb[] GetLightBulbs() => items.Select(x =>
x.item).OfType<LightBulb>().ToArray();


        private bool comparative_test_mode = false;
        private string prev_state = ″0″;
        private string cur_state = ″0″;

        public bool ComparativeTestMode
        {
            get => comparative_test_mode;
            set
            {
                comparative_test_mode = value;
                if (value)
                    prev_state = cur_state = Export();
            }
        }
```

```
public bool SomethingHasChanged => prev_state != cur_state;
```

```
    }
}
```

Файл Utils.cs

```
using System.Text;
using System.Collections.Generic;
using System.Linq;
using Avalonia.Controls;
using Avalonia.Media.Imaging;
using Avalonia;
using Avalonia.Media;
using System.Text.Json;
using LogicSimulator.ViewModels;
using System.Collections;
using System.Diagnostics;
using System;

using System.Data.SQLite;
using System.Data;
namespace LogicSimulator.Models {
    public static class Utils {

        public static string Base64Encode(string plainText)
        {
            var plainTextBytes = Encoding.UTF8.GetBytes(plainText);
            return System.Convert.ToBase64String(plainTextBytes);
        }
        public static string Base64Decode(string base64EncodedData)
        {

            var base64EncodedBytes =
System.Convert.FromBase64String(base64EncodedData);
            return Encoding.UTF8.GetString(base64EncodedBytes);
        }

        public static string JsonEscape(string str)
        {
            StringBuilder sb = new();
            foreach (char i in str)
            {
                sb.Append(i switch
                {
                    '”' => ”\\\””,
                    '\\' => ”\\\\”,
                    '$' => ”{$”,
                    _ => i
                });
            }
            return sb.ToString();
        }
        public static string Obj2json(object? obj)
        {
            switch (obj)
            {
                case null: return "null";
                case string @str: return '"' + JsonEscape(str) + '"';
                case bool @bool: return @bool ? "true" : "false";
                case short @short: return @short.ToString();
                case int @int: return @int.ToString();
                case long @long: return @long.ToString();
'.');           case float @float: return @float.ToString().Replace(',',
'.');           case double @double: return @double.ToString().Replace(',',

(int) @point.Y + '"';case Point @point: return "\"$p$" + (int) @point.X + "," +
(int) @size.Height + '"';case Size @size: return "\"$s$" + (int) @size.Width + "," +

                case Points @points: return "\"$P$" + string.Join("|",
@points.Select(p => (int) p.X + "," + (int) p.Y)) + '"';
'"';            case SolidColorBrush @color: return "\"$C$" + @color.Color +

                case Thickness @thickness: return "\"$T$" + @thickness.Left +
"," + @thickness.Top + "," + @thickness.Right + "," + @thickness.Bottom +
'"';

                case Dictionary<string, object?> @dict:
                {
                    StringBuilder sb = new();
                    sb.Append('{');
                    foreach (var entry in @dict)
                    {
                        if (sb.Length > 1)
                            sb.Append(", ");
                        sb.Append(Obj2json(entry.Key));
                        sb.Append(": ");
                        sb.Append(Obj2json(entry.Value));
                    }
                    sb.Append('}');
                    return sb.ToString();
                }
                case IEnumerable @list:
                {
                    StringBuilder sb = new();
                    sb.Append('[');
                    foreach (object? item in @list)
                    {
                        if (sb.Length > 1)
                            sb.Append(", ");
                        sb.Append(Obj2json(item));
                    }
                    sb.Append(']');
                    return sb.ToString();
                }
                default:
                    return "(" + obj.GetType() + " ???)";
            }
        }

        private static object JsonHandler(string str)
        {
            if (str.Length < 3 || str[0] != '$' || str[2] != '$')
                return str.Replace("{$", "$");
            string data = str[3..];

            string[] thick = str[1] == 'T' ? data.Split(',') :
System.Array.Empty<string>();
            return str[1] switch
            {
                'p' => Point.Parse(data),
                's' => Size.Parse(data),
                'C' => new SolidColorBrush(Color.Parse(data)),

                'T' => new Thickness(double.Parse(thick[0]),
double.Parse(thick[1]), double.Parse(thick[2]), double.Parse(thick[3])),
                _ => str,
            };
        }
        private static object? JsonHandler(object? obj)
        {
            if (obj == null)
                return null;

            if (obj is List<object?> @list)
                return @list.Select(JsonHandler).ToList();
            if (obj is Dictionary<string, object?> @dict)
            {

                return new Dictionary<string, object?>(@dict.Select(pair =>
new KeyValuePair<string, object?>(pair.Key, JsonHandler(pair.Value))));
            }
            if (obj is JsonElement @item)
            {
                switch (@item.ValueKind)
                {
                    case JsonValueKind.Undefined:
                        return null;
                    case JsonValueKind.Object:
                        Dictionary<string, object?> res = new();
                        foreach (var el in @item.EnumerateObject())
                            res[el.Name] = JsonHandler(el.Value);
                        return res;
                    case JsonValueKind.Array:

                        List<object?> res2 =
@item.EnumerateArray().Select(item => JsonHandler((object?) item)).ToList();
                        return res2;
                    case JsonValueKind.String:
                        var s = JsonHandler(@item.GetString() ?? "");
                        return s;
                    case JsonValueKind.Number:
                        if (@item.ToString().Contains('.'))
                            return @item.GetDouble();
                        long a = @item.GetInt64();
                        int b = @item.GetInt32();
                        if (a != b)
                            return a;
                        return b;
                    case JsonValueKind.True: return true;
                    case JsonValueKind.False: return false;
                    case JsonValueKind.Null: return null;
                }
            }
            Log.Write("JT: " + obj.GetType());

            return obj;
        }
        public static object? Json2obj(string json)
        {
            json = json.Trim();
            if (json.Length == 0)
                return null;

            object? data;
            if (json[0] == '[')
                data = JsonSerializer.Deserialize<List<object?>>(json);
            else if (json[0] == '{')
object?>>(json);data = JsonSerializer.Deserialize<Dictionary<string,
            else
                return null;

            return JsonHandler(data);
        }

        private static string ToJSONHandler(string str)
        {
'0')        if (str.Length > 1 && str[0] == '$' && str[1] <= '9' && str[1] >=
                return str[1..];
            str = str.Replace("\\", "\\\\");
            return str switch
            {
                "null" => "null",
                "undefined" => "undefined",
                "_BOOL_yeah" => "true",
                "_BOOL_nop" => "false",
                _ => '"' + str + '"',
            };
        }

        public static void RenderToFile(Control target, string path)
        {
            double w = target.Bounds.Width, h = target.Bounds.Height;
            var pixelSize = new PixelSize((int) w, (int) h);
            var size = new Size(w, h);
            using RenderTargetBitmap bitmap = new(pixelSize);
            target.Measure(size);
            target.Arrange(new Rect(size));
            bitmap.Render(target);
            bitmap.Save(path);
        }

        public static string TrimAll(this string str)
        {
            StringBuilder sb = new();
```

```
for (int i = 0; i < str.Length; i++)
```

```
            {
                if (i > 0 && str[i] == ’ ’ && str[i - 1] == ’ ’)
                    continue;
                sb.Append(str[i]);
            }
            return sb.ToString().Trim();
        }


        public static string[] NormSplit(this string str) =>
str.TrimAll().Split(’ ’);

        public static string GetStackInfo()
        {
            var st = new StackTrace();
            var sb = new StringBuilder();
            for (int i = 1; i < 11; i++)
            {
                var frame = st.GetFrame(i);
                if (frame == null)
                    continue;

                var method = frame.GetMethod();
                if (method == null || method.ReflectedType == null)
                    continue;

″);             sb.Append(method.ReflectedType.Name + ″ ″ + method.Name + ″ |
                if (i == 5)
                    sb.Append(″\n    ″);
            }
            return sb.ToString();
        }

        public static int Normalize(this int num, int min, int max)
        {
            if (num < min)
                return min;
            if (num > max)
                return max;
            return num;
        }
max)    public static double Normalize(this double num, double min, double
        {
            if (num < min)
                return min;
            if (num > max)
                return max;
            return num;
        }

        public static double Hypot(this Point delta)
        {
            return Math.Sqrt(Math.Pow(delta.X, 2) + Math.Pow(delta.Y, 2));
        }
        public static double Hypot(this Point A, Point B)
        {
```

```
Point delta = A - B;
```

```
            return Math.Sqrt(Math.Pow(delta.X, 2) + Math.Pow(delta.Y, 2));
        }

        public static double? ToDouble(this object num)
        {
            return num switch
            {
                int @int => @int,
                long @long => @long,
                double @double => @double,
                _ => null,
            };
        }

        public static int Min(this int A, int B) => A < B ? A : B;
        public static int Max(this int A, int B) => A > B ? A : B;
        public static double Min(this double A, double B) => A < B ? A : B;
        public static double Max(this double A, double B) => A > B ? A : B;

        public static void Remove(this Control item)
        {
            var p = (Panel?) item.Parent;
            p?.Children.Remove(item);
        }

        public static Point Center(this Visual item, Visual? parent)
        {
            var tb = item.TransformedBounds;
            if (tb == null)
                return new();
            var bounds = tb.Value.Bounds.TransformToAABB(tb.Value.Transform);
            var res = bounds.Center;
            if (parent == null)
                return res;

            var tb2 = parent.TransformedBounds;
            if (tb2 == null)
                return res;

            var bounds2 =
tb2.Value.Bounds.TransformToAABB(tb2.Value.Transform);
            return res - bounds2.TopLeft;
        }


        public static DateTime UnixTimeStampToDateTime(this long
unixTimeStamp)
        {

            DateTime dateTime = new(1970, 1, 1, 0, 0, 0, 0,
DateTimeKind.Utc);
            dateTime = dateTime.AddSeconds(unixTimeStamp).ToLocalTime();
            return dateTime;
        }
        public static string UnixTimeStampToString(this long unixTimeStamp)
        {
```

```
UnixTimeStampToDateTime(unixTimeStamp).ToString("yyyy/MM/dd H:mm:ss");
```

```
        }

        internal static void Obj2sqlite_proj_list(string[] proj_list, string
path)
        {
            using var con = new SQLiteConnection("Data Source=" + path);

            con.Open();
            if (con.State != ConnectionState.Open)
            {
                Log.Write("Failed to open SQLite: " + con.State);
                return;
            }

            string comm = @"
CREATE TABLE header (
    num       INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    path      TEXT    NOT NULL
);";
            new SQLiteCommand(comm, con).ExecuteReader().Close();

            var arr = proj_list.Select(x => $"('{x}')");
            comm = $"INSERT INTO header (path) VALUES {string.Join(", ",
arr)};";
            new SQLiteCommand(comm, con).ExecuteReader().Close();

            con.Dispose();
        }

        internal static string[] SQLite_proj_list2obj(string path) {
            using var con = new SQLiteConnection("Data Source=" + path);
            con.Open();
            if (con.State != ConnectionState.Open)
                throw new Exception("Failed to open SQLite: " + con.State);

            List<string> res = new();

            var sql_comm = new SQLiteCommand("SELECT * FROM header", con);
            using (var reader = sql_comm.ExecuteReader()) {
                if (!reader.HasRows)
                    throw new Exception("Failed to read SQLite header
table");
                if (!reader.Read())
                    throw new Exception("Header table is empty");


                var row = Enumerable.Range(0,
reader.VisibleFieldCount).Select(x => reader[x]).ToArray();
                Log.Write("row: " + Obj2json(row));

                res.Add((string) row[1]);
            }

            con.Dispose();
            return res.ToArray();
        }
    }
}
```

Файл 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();
            });
            Open = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncOpen();
                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();
            CurrentProj = newy;
            mw.Show();
            mw.Update();
            me?.Close();
        }
        void FuncOpen()
        {
            if (me == null)
                return;

            var selected = map.filer.SelectProjectFile(me);
            if (selected == null)
                return;

            CurrentProj = selected;
            mw.Show();
            mw.Update();
            me?.Close();
        }
        void FuncExit()
        {
            me?.Close();
            mw.Close();
        }

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

            CurrentProj = proj;

            mw.Show();
            mw.Update();
            me?.Close();
        }
        public static MainWindow GetMW => mw;
    }
}
```

Файл MainWindowViewModel.cs

```
using Avalonia.Controls;
using Avalonia.Controls.Presenters;
using Avalonia.Input;
using LogicSimulator.Models;
using LogicSimulator.Views;
using LogicSimulator.Views.Shapes;
using ReactiveUI;
using System.Collections.Generic;
using System.Collections.ObjectModel;
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 > 45)
                    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
    {
        private string log = "";
        public string Logg { get => log; set => this.RaiseAndSetIfChanged(ref
log, value); }

        public MainWindowViewModel()
        {
            Log.Mwvm = this;
            Comm = ReactiveCommand.Create<string, Unit>(n =>
            {
                FuncComm(n);
                return new Unit();
            });
            NewItem = ReactiveCommand.Create<Unit, Unit>(_ =>
            {
                FuncNewItem();
                return new Unit();
            });
        }

        private Window? mw;
        public void AddWindow(Window window)
        {
            var canv = window.Find<Canvas>("Canvas");

            mw = window;
            map.canv = canv;
            if (canv == null)
                return; // Такого не бывает

            canv.Children.Add(map.Marker);
            canv.Children.Add(map.Marker2);

            var panel = (Panel?) canv.Parent;
            if (panel == null)
                return; // Такого не бывает

e) =>       panel.PointerPressed += (object? sender, PointerPressedEventArgs
            {
                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();
                        newy.Move(pos);
                        map.AddItem(newy);
                    }
                }
            };

            panel.PointerWheelChanged += (object? sender,
PointerWheelEventArgs e) =>
            {
                if (e.Source != null && e.Source is Control @control)
```

```
                    map.WheelMove(@control, e.Delta.Y,
e.GetCurrentPoint(canv).Position);
                };
                mw.KeyDown += (object? sender, KeyEventArgs e) =>
                {
                    if (e.Source != null && e.Source is Control @control)
                        map.KeyPressed(@control, e.Key);
                };
        }

        public static IGate[] ItemTypes { get => map.item_types; }

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

        Grid? cur_grid;
        TextBlock? old_b_child;
        object? old_b_child_tag;
        string? prev_scheme_name;


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


        public static ObservableCollection<Scheme> Schemes { get =>
CurrentProj == null ? new() : CurrentProj.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 Grid g2)
                src = g2;
            if (src is Grid g3 && g3.Children[0] is TextBlock tb2)
                src = tb2;

            if (src is not TextBlock tb)
                return;

            var p = tb.Parent;
            if (p == null)
                return;

            if (old_b_child != null)
                if (cur_grid != null)
                    cur_grid.Children[0] = old_b_child;

            if (p is not Grid g)
                return;
            cur_grid = g;
            old_b_child = tb;
            old_b_child_tag = tb.Tag;
            prev_scheme_name = tb.Text;

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

            cur_grid.Children[0] = newy;

            newy.KeyUp += (object? sender, KeyEventArgs e) =>
            {
                if (e.Key != Key.Return)
                    return;

                if (newy.Text != prev_scheme_name)
                {
                    if ((string?) tb.Tag == "p_name")
                        CurrentProj?.ChangeName(newy.Text);
                    else if

                        (old_b_child_tag is Scheme scheme)
scheme.ChangeName(newy.Text);
                }

                cur_grid.Children[0] = tb;
                cur_grid = null; old_b_child = null;
            };
        }

        public void Update()
        {
            map.ImportScheme();

            this.RaisePropertyChanged(new(nameof(ProjName)));
            this.RaisePropertyChanged(new(nameof(Schemes)));
            this.RaisePropertyChanged(new(nameof(CanSave)));
            if (mw != null) mw.Width++;
        }


        public static bool CanSave { get => CurrentProj != null &&
CurrentProj.CanSave(); }

        public void FuncComm(string Comm)
        {
            switch (Comm)
            {
                case "Create":
                    var newy = map.filer.CreateProject();
                    CurrentProj = newy;
                    Update();
                    break;
                case "Open":
                    if (mw == null)
                        break;
                    var selected = map.filer.SelectProjectFile(mw);
                    if (selected != null)
```

{

```
                        CurrentProj = selected;
                        Update();
                    }
                    break;
                case "Save":
                    map.Export();

                    File.WriteAllText("../../../for_test.json",
Utils.Obj2json((map.current_scheme ?? throw new System.Exception("Unknown
error")).Export()));
                    break;
                case "SaveAs":
                    map.Export();
                    if (mw != null)
                        CurrentProj?.SaveAs(mw);
                    this.RaisePropertyChanged(new(nameof(CanSave)));
                    break;
                case "ExitToLauncher":
                    new LauncherWindow().Show();
                    mw?.Hide();
                    break;
                case "Exit":
                    mw?.Close();
                    break;
            }
        }

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

        private static void FuncNewItem()
        {
            CurrentProj?.AddScheme(null);
        }

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


        public static bool LockSelfConnect { get => map.lock_self_connect;
set => map.lock_self_connect = value; }
    }
}
```

Файл ViewModelBase.cs

```
using LogicSimulator.Models;
using ReactiveUI;

namespace LogicSimulator.ViewModels
{
    public class ViewModelBase: ReactiveObject
    {
        public readonly static Mapper map = new();
        private static Project? current_proj;
        protected static Project? CurrentProj
        {
            get => current_proj;
            set
            {
                if (value == null) return;
                current_proj = value;
                map.current_scheme = value.GetFirstScheme();
            }
        }

        public static Project? TopSecretGetProj() => current_proj;
    }
}
```

Файл AND_2.axaml.cs

```
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 UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0, 0 },
            new int[] { 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();


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

Файл AND_8.axaml.cs

```
using Avalonia.Controls;
using System.ComponentModel;
```

```
namespace LogicSimulator.Views.Shapes
```

```
{
    public partial class AND_8: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 11;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0, 0, 0, 0, 0, 0, 0, 0 },
            new int[] { 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();


        public void Brain(ref bool[] ins, ref bool[] outs) => outs[0] =
ins[0] && ins[1] && ins[2] && ins[3] && ins[3] && ins[5] && ins[6] && ins[7];
    }
}
```

Файл Button.axaml.cs

```
using Avalonia.Controls;
using Avalonia.Controls.Shapes;
using Avalonia.Input;
using Avalonia.Media;
using LogicSimulator.Models;
using System.ComponentModel;

namespace LogicSimulator.Views.Shapes
{
    public partial class Button: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 6;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            System.Array.Empty<int>(),
            new int[] { 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();

5.5;    public double ButtonSize => width.Min(height) - BodyStrokeSize.Left *

        bool my_state = false;
        private void Press(object? sender, PointerPressedEventArgs e)
        {
            if (e.Source is not Ellipse button)
                return;
            my_state = true;
            button.Fill = new SolidColorBrush(Color.Parse("#e50000"));
        }
        private void Release(object? sender, PointerReleasedEventArgs e)
        {
            if (e.Source is not Ellipse button)
                return;
            my_state = false;
            button.Fill = new SolidColorBrush(Color.Parse("#ff0000"));
        }


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

Файл FlipFlop.axaml.cs

```
using Avalonia.Controls;
using LogicSimulator.ViewModels;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;

namespace LogicSimulator.Views.Shapes
{
    public partial class FlipFlop: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 9;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0, 0, 0 },
            new int[] { 1, 1, 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();

        private readonly bool[] prev = new bool[3];
        private readonly bool[] out_d = new bool[3];

        public void Brain(ref bool[] ins, ref bool[] outs)
        {
            for (int i = 0; i < 3; i++)
            {
                if (prev[i] && !ins[i])
                    out_d[i] = !out_d[i];
                outs[i] = out_d[i];
                prev[i] = ins[i];
            }
        }

        public override Dictionary<string, object> ExtraExport() =>
            new()
            {
+ "." +         ["state"] = string.Join('.', prev.Select(x => x ? '1' : '0'))
                string.Join('.', out_d.Select(x => x ? '1' : '0'))
            };

        public override void ExtraImport(string key, object extra)
        {
            if (key != "state")
            {
                Log.Write(key + "-invalid element state record type");
                return;
            }
            if (extra is not string @state) {
                Log.Write("Invalid element state record type: " + extra);
                return;
            }
            var arr = @state.Split('.');
            prev[0] = arr[0] == "1";
            prev[1] = arr[1] == "1";
            prev[2] = arr[2] == "1";
            out_d[0] = arr[3] == "1";
            out_d[1] = arr[4] == "1";
            out_d[2] = arr[5] == "1";
        }
    }
}
```

Файл 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;
using System.Linq;

namespace LogicSimulator.Views.Shapes
{
    public abstract class GateBase: UserControl
    {
        public int CountIns { get; private set; }
        public int CountOuts { get; private set; }
        public abstract UserControl GetSelf();
        protected abstract IGate GetSelfI { get; }
        protected abstract void Init();
        protected abstract int[][] Sides { get; }

        protected readonly Line[] line_arr;
        protected readonly Ellipse[] pins;
        protected readonly Border border;

        protected bool use_top;
        protected bool use_left;
        protected bool use_right;
        protected bool use_bottom;
        private int[][] pin_data;

        public GateBase()
        {
            var sides = Sides;
            use_top = sides[0].Length > 0;
            use_left = sides[1].Length > 0;
            use_right = sides[2].Length > 0;
            use_bottom = sides[3].Length > 0;
            int ins = 0, outs = 0, ios = 0, n = 0;
            List<int[]> pin_d = new();
            foreach (var side in sides)
                foreach (var type in side)
                {
                    switch (type)
                    {
break;                  case 0: pin_d.Add(new int[] { 0, ins }); ins++;
break;                  case 1: pin_d.Add(new int[] { 1, outs }); outs++;
break;                  case 2: pin_d.Add(new int[] { 2, ios }); ios++;
                    }
                    if (type != -1)
                        n++;
                }
            pin_data = pin_d.ToArray();
            CountIns = ins;
            CountOuts = outs + ios;

            width = MinW; height = MinH;
            if (height < width)
                height = width;
            DataContext = GetSelf();
            Init();

            var canv = (Canvas) LogicalChildren[0];
            List<Line> list = new();
            List<Ellipse> list2 = new();
            if (canv.Children[0] is not Border b)
                throw new Exception("Unexpected error");
            border = b;
```

```
border.ZIndex = 2;
```

```
            foreach (var side in sides)
                foreach (var type in side)
                {
                    if (type < 0)
                        continue;

                    var newy = new Line() { Tag = "Pin", ZIndex = 1, Stroke = 
Brushes.Gray };
                    list.Add(newy);
                    canv.Children.Add(newy);


                    var newy2 = new Ellipse() { Tag = type == 0 ? "In" : type 
== 1 ? "Out" : "IO", ZIndex = 2, Stroke = Brushes.Gray, Fill = new 
SolidColorBrush(Color.Parse("#0000")) };
                    list2.Add(newy2);
                    canv.Children.Add(newy2);
                }
            line_arr = list.ToArray();
            pins = list2.ToArray();

            joins_in = new JoinedItems?[ins];
            joins_out = new List<JoinedItems>[outs];
            for (int i = 0; i < outs; i++)
                joins_out[i] = new();

            MyRecalcSizes();
        }

        public void Move(Point pos, bool global = false)
        {
            Margin = new(pos.X - UC_Width / 2, pos.Y - UC_Height / 2, 0, 0);
            UpdateJoins(global);
        }


        private double MinW => BodyRadius.TopLeft * 1.5 + (EllipseSize + 
BaseFraction * 2) * (Sides[0].Length.Max(Sides[3].Length).Max(2) - 0.8);

        private double MinH => BodyRadius.TopLeft * 1.5 + (EllipseSize + 
BaseFraction * 2) * (Sides[1].Length.Max(Sides[2].Length).Max(2) - 0.8);
        public void Resize(Size size, bool global = false)
        {
            width = global ? size.Width : size.Width.Max(MinW);
            height = global ? size.Height : size.Height.Max(MinH);
            RecalcSizes();
            UpdateJoins(global);
        }
        public void ChangeScale(double scale, bool global = false)
        {
            var fix = GetPos();
            base_size *= scale;
            width *= scale;
            height *= scale;
            Move(fix, global);
            RecalcSizes();
```

```
UpdateJoins(global);
```

```
        }

        public Point GetPos() => new(Margin.Left + UC_Width / 2, Margin.Top +
UC_Height / 2);
        public Size GetSize() => new(Width, Height);
        public Size GetBodySize() => new(width, height);

        private Point pose;
        public void SavePose() => pose = GetPos();
        public Point GetPose() => pose;
        public Rect GetBounds() => new(Margin.Left, Margin.Top, UC_Width,
UC_Height);

        protected 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(use_left ? base_size : 0, use_top
? 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 => (use_left ? base_size : 0) + width +
(use_right ? base_size : 0);

        public double UC_Height => (use_top ? base_size : 0) + height +
(use_bottom ? base_size : 0);

        public double FontSizze => BodyRadius.TopLeft / 1.3;

        public Thickness ImageMargins
        {
            get
            {
                double R = BodyRadius.BottomLeft;
                double num = R - R / Math.Sqrt(2);
                return new(0, 0, num, num);
            }
        }

        public Point[][] PinPoints
        {
            get
            {
                List<Point[]> res = new();
```

```
i
n
t

n

=

-
1
;
```

```
                double R = BodyRadius.TopLeft;
                double min = EllipseSize + BaseFraction * 2;
                double pin_start = EllipseSize - EllipseStrokeSize / 2;
                double pin_width = base_size - EllipseSize + PinStrokeSize;
                foreach (var side in Sides)
                {
                    n++;
                    double count = side.Length;
                    if (count == 0)
                        continue;

                    double body_len = n == 0 || n == 3 ? height : width;

                    double body_len2 = n == 0 || n == 3 ? width : height;

                    double delta = n < 2 ? pin_start : (n == 2 ? (use_left ?
base_size : 0) : (use_top ? base_size : 0)) + body_len - EllipseStrokeSize /
2;
R;                  double left = R, mid = body_len2 / 2, right = body_len2 -
min;                bool overflow = count > 1 && (right - left) / count <
                    int n2 = 0;
                    foreach (int type in side)
                    {
                        double delta2 = overflow ?
                            mid + min * (n2 - (count - 1) / 2) :
1);                         left + (right - left) / (count * 2) * (n2 * 2 +
                        if (type >= 0)
                            res.Add(n == 0 || n == 3 ?

                            new Point[] { new(delta2 + (use_left ? base_size
: 0), delta), new(0, pin_width) } :

                            new Point[] { new(delta, delta2 + (use_top ?
base_size : 0)), new(pin_width, 0) });
                        n2++;
                    }
                }
                return res.ToArray();
            }
        }

        public Thickness[] EllipseMargins
        {
            get
            {
                Point[][] pins = PinPoints;
                double R2 = EllipseSize / 2;
                double X = UC_Width - EllipseSize;
                double Y = UC_Height - EllipseSize;
                int n = 0, side_n = 0;
                List<Thickness> list = new();
                foreach (var side in Sides)
                {
                    foreach (var type in side)
                    {
                        if (type == -1)
```

```
                                   continue;
```

```
                        var pin_line = pins[n++];
                        switch (side_n)
                        {
                            case 0: list.Add(new(pin_line[0].Y - R2, 0, 0, 0)); break;
                            case 1: list.Add(new(0, pin_line[0].Y - R2, 0, 0)); break;
                            case 2: list.Add(new(X, pin_line[0].Y - R2, 0, 0)); break;
                            case 3: list.Add(new(pin_line[0].X - R2, Y, 0, 0)); break;
                        }
                    }
                    side_n++;
                }
                return ellipse_margins = list.ToArray();
            }
        }

        public double ImageSize => base_size / 25 * 24;



#pragma warning disable CS0108
        public event PropertyChangedEventHandler? PropertyChanged;
#pragma warning restore CS0108

        protected void RecalcSizes()
        {
            PropertyChanged?.Invoke(this, new(nameof(BodyStrokeSize)));
            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(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(nameof(ImageSize)));

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

            MyRecalcSizes();
        }

        protected void MyRecalcSizes()
        {
            var pin_points = PinPoints;
            var pin_stroke_size = PinStrokeSize;
            int n = 0;
            foreach (var line in line_arr)
            {
                var A = pin_points[n][0];
                var B = pin_points[n++][1];

                line.StrokeThickness = pin_stroke_size;
                line.Margin = new(A.X, A.Y, 0, 0);
                line.EndPoint = B;
            }

            n = 0;
            var ellipse_margin = EllipseMargins;
            var ellipse_size = EllipseSize;
            var ellipse_stroke_size = EllipseStrokeSize;
            foreach (var pin in pins)
            {
                pin.Margin = ellipse_margin[n++];
                pin.Width = ellipse_size;
                pin.Height = ellipse_size;
                pin.StrokeThickness = ellipse_stroke_size;
            }
        }

        protected JoinedItems?[] joins_in;
        protected List<JoinedItems>[] joins_out;

        public void AddJoin(JoinedItems join)
        {
            for (int i = 0; i < 2; i++)
            {
                var dist = i == 0 ? join.A : join.B;
                if (dist.parent == this)
                {
                    int[] data = pin_data[dist.num];
                    int n = data[1];
                    if (data[0] == 0)
                    {
                        joins_in[n]?.Delete();
                        joins_in[n] = join;
                    }
                    else
                    {
                        joins_out[n].Add(join);
                    }
                }
            }
            skip_upd = false;
        }

        public void RemoveJoin(JoinedItems join)
        {
            for (int i = 0; i < 2; i++)
            {
                var dist = i == 0 ? join.A : join.B;
                if (dist.parent == this)
                {
                    int[] data = pin_data[dist.num];
                    int n = data[1];
                    if (data[0] == 0)
                        joins_in[n] = null;
                    else joins_out[n].Remove(join);
                }
            }
            skip_upd = false;
        }

        public void UpdateJoins(bool global)
        {
            foreach (var join in joins_in)
                join?.Update();
            if (!global)
                foreach (var joins in joins_out)
                    foreach (var join in joins)
                        join.Update();
        }

        public void ClearJoins()
        {
            foreach (var join in joins_in)
                join?.Delete();
            foreach (var joins in joins_out)
                foreach (var join in joins.ToArray())
                    join.Delete();
        }

        public void SetJoinColor(int o_num, bool value)
        {
            var joins = joins_out[o_num];
            Dispatcher.UIThread.InvokeAsync(() =>
            {
                foreach(var join in joins)
Brushes.DarkGray;   join.line.Stroke = value ? Brushes.Lime :
            });
        }

        public bool ContainsJoin(JoinedItems join)
        {
            foreach (var join2 in joins_in)
                if (join == join2)
                    return true;
            foreach (var joins in joins_out)
                foreach (var join2 in joins)
                    if (join == join2)
                        return true;
            return false;
        }

        public Distantor GetPin(Ellipse finded)
        {
            int n = 0;
            foreach (var pin in pins)
            {
                if (pin == finded)
                    return new(GetSelfI, n, (string?) finded.Tag ?? "");
                n++;
            }
            throw new Exception("Unexpected error");
        }

        Thickness[] ellipse_margins = Array.Empty<Thickness>();
        public Point GetPinPos(int n)
        {
            var m = ellipse_margins[n];
            double R2 = EllipseSize / 2;
R2);        return new Point(Margin.Left + m.Left + R2, Margin.Top + m.Top +
        }

        public int[][] GetPinData() => pin_data;

        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_in[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];
                        int[] data = p.GetPinData()[item.num];
                        me.ins[i] = meta.outs[data[1]];
                    }
                }
                if (join.B.parent == this)
                {
                    var item = join.A;
                    if (item.tag == "Out" || item.tag == "IO")
                    {
                        var p = item.parent;
                        Meta meta = ids[p];
                        int[] data = p.GetPinData()[item.num];
                        me.ins[i] = meta.outs[data[1]];
                    }
                }
            }
        }

        public abstract int TypeId { get; }

        public object Export()
        {
            var res = new Dictionary<string, object>
            {
                ["id"] = TypeId,
                ["pos"] = GetPos(),
                ["size"] = GetBodySize(),
                ["base_size"] = base_size
            };
            var res2 = ExtraExport();
            if (res2 != null)
                foreach (var item in res2)
                    res.Add(item.Key, item.Value);
            return res;
        }
        public virtual Dictionary<string, object>? ExtraExport() => null;

        public List<object[]> ExportJoins(Dictionary<IGate, int> to_num)
        {
            List<object[]> res = new();
            foreach (var joins in joins_out)
                foreach (var join in joins)
                {
                    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 void Import(Dictionary<string, object> dict)
        {
            double new_b_size = base_size;
            Point new_pos = GetPos();
            Size new_size = GetSize();
            foreach (var item in dict)
            {
                object value = item.Value;
                switch (item.Key)
                {
                    case "id":
                        if (value is int @id)
                        {
                            if (@id != TypeId)
                                throw new ArgumentException("Invalid id: " +
@id + " Expected: " + TypeId);
                        }
                        else
                        {
                            Log.Write("Wrong element record id type: " +
value);
                        }
                        break;
                    case "pos":
                        if (value is Point @pos)
                            new_pos = @pos;
                        else
                            Log.Write("Invalid element pos record type: " +
value);
                        break;
                    case "base_size":
                        double? b_size = value.ToDouble();
                        if (b_size != null)
                            new_b_size = (double) b_size;
                        else
" + value);                 Log.Write("Invalid element base_size record type:
                        break;
                    case "size":
                        if (value is Size @size)
                            new_size = @size;
                        else
value);                     Log.Write("Invalid element size record type: " +
                        break;
                    default:
                        ExtraImport(item.Key, value);
                        break;
                }
            }
            base_size = new_b_size;
            Resize(new_size, true);
            Move(new_pos);
        }
        public virtual void ExtraImport(string key, object extra)
        {
            Log.Write(key + "-item entry is not supported");
        }

        public Ellipse SecretGetPin(int n) => pins[n];
    }
}
```

Файл 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, bool global = false);
        public void Resize(Size size, bool global = false);
        public void ChangeScale(double scale, bool global = false);
        public void SavePose();
        public Point GetPose();
        public Rect GetBounds();

        public Distantor GetPin(Ellipse finded);
        public Point GetPinPos(int n);

        public void AddJoin(JoinedItems join);
        public void RemoveJoin(JoinedItems join);
        public void ClearJoins();
        public void SetJoinColor(int o_num, bool value);
        public bool ContainsJoin(JoinedItems join);

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

        public Ellipse SecretGetPin(int n);
    }
}
```

Файл LightBulb.axaml.cs

```
using Avalonia.Controls;
using Avalonia.Media;
using Avalonia.Threading;
using LogicSimulator.ViewModels;
using System;
using System.Collections.Generic;
using System.ComponentModel;

namespace LogicSimulator.Views.Shapes
{
    public partial class LightBulb: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 7;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            Array.Empty<int>(),
            new int[] { 0 },
            Array.Empty<int>(),
            Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();
        readonly SolidColorBrush ColorA = new(Color.Parse("#00ff00")); // On
        readonly SolidColorBrush ColorB = new(Color.Parse("#1c1c1c")); // Off
        public void Brain(ref bool[] ins, ref bool[] outs)
        {
            var value = state = ins[0];
            Dispatcher.UIThread.InvokeAsync(() =>
            {
                border.Background = value ? ColorA : ColorB;
            });
        }

        bool state;

        public bool GetState() => state;

        public override Dictionary<string, object> ExtraExport() => new() {
["state"] = state };

        public override void ExtraImport(string key, object extra)
        {
            if (key != "state")
            {
                Log.Write(key + "-item entry is not supported");
                return;
            }
            if (extra is not bool @st)
            {
                Log.Write("Invalid element state record type: " + extra);
                return;
            }
            state = @st;
            if (state)
                border.Background = ColorA;
        }
    }
}
```

Файл NAND_2.axaml.cs

```
using Avalonia;
using Avalonia.Controls;
using System.ComponentModel;

namespace LogicSimulator.Views.Shapes
{
    public partial class NAND_2: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 8;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0, 0 },
            new int[] { 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();

        public double InvertorSize => EllipseSize / 2;
        public double InvertorStrokeSize => EllipseStrokeSize / 2;

        public Thickness InvertorMargin => new(width + BaseFraction * 2 -
InvertorSize / 2, 0, 0, 0);


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

Файл NOT.axaml.cs

```
using Avalonia.Controls;
using System.ComponentModel;

namespace LogicSimulator.Views.Shapes
{
    public partial class NOT: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 2;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0 },
            new int[] { 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();

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

Файл OR_2.axaml.cs

```
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 UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0, 0 },
            new int[] { 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();


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

Файл OR_8.axaml.cs

```
using Avalonia.Controls;
using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {
    public partial class OR_8: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 10;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0, 0, 0, 0, 0, 0, 0, 0 },
            new int[] { 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();
        public void Brain(ref bool[] ins, ref bool[] outs) => outs[0] =
ins[0] || ins[1] || ins[2] || ins[3] || ins[4] || ins[5] || ins[6] || ins[7];
    }
}
```

Файл SuM.axaml.cs

```
using Avalonia.Controls;
using System.ComponentModel;

namespace LogicSimulator.Views.Shapes {
    public partial class SuM: GateBase, IGate, INotifyPropertyChanged
    {
        public override int TypeId => 4;

        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            System.Array.Empty<int>(),
            new int[] { 0, 0 },
            new int[] { 1, 1 },
            System.Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();

        public void Brain(ref bool[] ins, ref bool[] outs)
        {
            outs[0] = (ins[0] | ins[1]) & !(ins[0] & ins[1]);
            outs[1] = ins[0] & ins[1];
        }
    }
}
```

Файл 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 UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;
        protected override int[][] Sides => new int[][]
        {
            Array.Empty<int>(),
            Array.Empty<int>(),
            new int[] { 1 },
            Array.Empty<int>()
        };

        protected override void Init() => InitializeComponent();


        bool my_state = false;
        Point? press_pos;

        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 ?
"#e50000" : "#ff0000"));
        }

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


        public override Dictionary<string, object> ExtraExport() => new() {
["state"] = my_state };

        public override void ExtraImport(string key, object extra)
        {
            if (key != "state")
            {
```

```
        Log.Write(key + ″-item entry is not supported″);
```

```
                return;
            }
            if (extra is not bool @state)
            {
                Log.Write("Invalid element state record type: " + extra);
                return;
            }
            my_state = @state;
            if (my_state)

                border.Background = new
SolidColorBrush(Color.Parse("#7d1414"));
        }

        public void SetState(bool state)
        {
            my_state = state;

            border.Background = new SolidColorBrush(Color.Parse(state ?
"#7d1414" : "#d32f2e"));
        }
    }
}
```

Файл LauncherWindow.axaml.cs

```
using Avalonia.Controls;
using LogicSimulator.ViewModels;

namespace LogicSimulator.Views
{
    public partial class LauncherWindow: Window
    {
        readonly LauncherWindowViewModel lwvm;

        public LauncherWindow()
        {
            InitializeComponent();
            lwvm = new LauncherWindowViewModel();
            DataContext = lwvm;
            lwvm.AddWindow(this);
        }


        public void DTapped(object? sender,
Avalonia.Interactivity.RoutedEventArgs e)
        {
            lwvm.DTapped(sender, e);
        }
    }
}
```

Файл MainWindow.axaml.cs

```
using Avalonia.Controls;
using LogicSimulator.ViewModels;

namespace LogicSimulator.Views {
    public partial class MainWindow: Window
    {
        readonly MainWindowViewModel mwvm;

        public MainWindow()
        {
            InitializeComponent();
            mwvm = new MainWindowViewModel();
            DataContext = mwvm;
            mwvm.AddWindow(this);
        }


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

        public void Update() => mwvm.Update();
    }
}
```

Файл App.axaml.cs

```
using Avalonia;
using Avalonia.Controls.ApplicationLifetimes;
using Avalonia.Markup.Xaml;
using LogicSimulator.Views;
using System.IO;

namespace LogicSimulator {
    public partial class App: Application
    {
        public override void Initialize()
        {
            AvaloniaXamlLoader.Load(this);
        }

        public override void OnFrameworkInitializationCompleted()
        {

            if (ApplicationLifetime is
IClassicDesktopStyleApplicationLifetime desktop)
                desktop.MainWindow = new LauncherWindow();
            base.OnFrameworkInitializationCompleted();
            IncrementBuildNum();
        }

        private static void IncrementBuildNum()
        {
            if (lock_inc_build) return;

            string path = "../../../../build.num";
            int num;
            try
            {
                num = int.Parse(File.ReadAllText(path));
            }
            catch (FileNotFoundException)
            {
                num = 0;
            }
            num++;
            File.WriteAllText(path, num.ToString());
        }

        public static bool lock_inc_build = false;
    }
}
```

Файл Program.cs

```
using Avalonia;
using Avalonia.ReactiveUI;
using System;

namespace LogicSimulator {
    public class Program {
any     // Initialization code. Don't use any Avalonia, third-party APIs or

        // SynchronizationContext-reliant code before AppMain is called:
things aren't initialized
        // yet and stuff might break.
        [STAThread]
        public static void Main(string[] args) => BuildAvaloniaApp()
            .StartWithClassicDesktopLifetime(args);

designer// Avalonia configuration, don't remove; also used by visual
        public static AppBuilder BuildAvaloniaApp()
            => AppBuilder.Configure<App>()
                .UsePlatformDetect()
                .LogToTrace()
                .UseReactiveUI();
    }
}
```

Файл ViewLocators.cs

```
using Avalonia.Controls;
using Avalonia.Controls.Templates;
using LogicSimulator.ViewModels;
using System;

namespace LogicSimulator
{
    public class ViewLocator: IDataTemplate
    {
        public IControl Build(object data)
        {
            var name = data.GetType().FullName!.Replace("ViewModel", "View");
            var type = Type.GetType(name);

            if (type != null)
            {
                return (Control) Activator.CreateInstance(type)!;
            }

            return new TextBlock { Text = "Not Found: " + name };
        }

        public bool Match(object data)
        {
            return data is ViewModelBase;
        }
    }
}
```