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

**Расчётно-графическая работа**

Дисциплина: Визуальное программирование и человеко-машинное взаимодействие

Тема: «Приложение-симулятор логических схем»
Вариант 21

Выполнил:

Студент группы ИП-117

Чероченко Д.А.

Проверил:

Ассистент кафедры ПМ и К

Меркулов И. А.

Новосибирск 2023

## Содержание

## 1. Задание

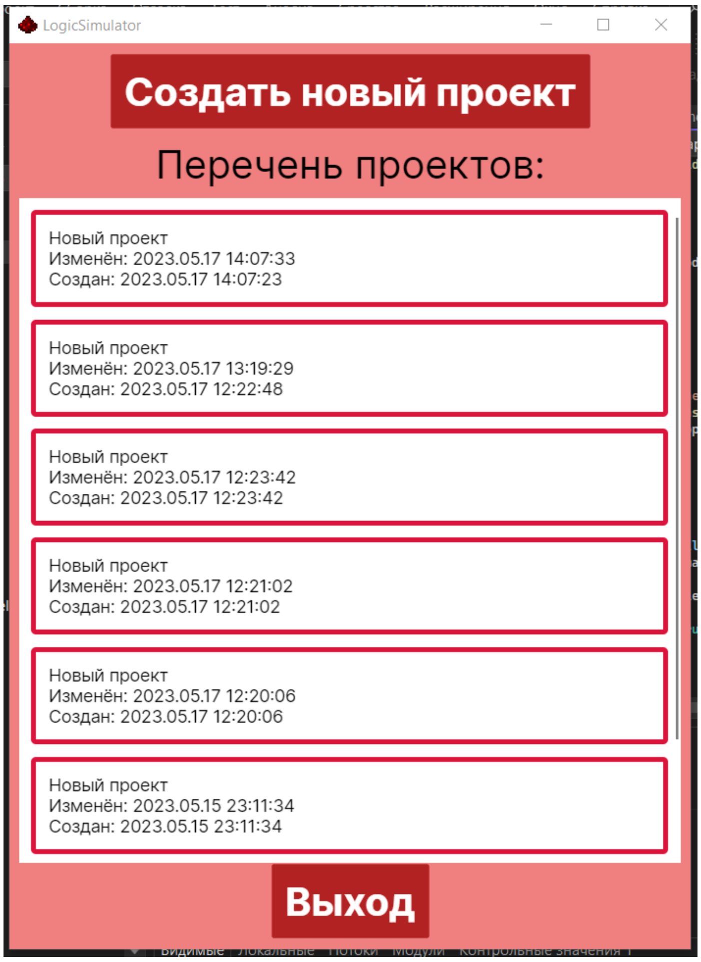
Реализовать приложение-симулятор логических схем.

Работа состоит из следующих этапов:

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

## 2. Пример работы программы



LogicSimulator

Создать новый проект

Перечень проектов:

Новый проект
Изменён: 2023.05.17 14:07:33
Создан: 2023.05.17 14:07:23

Новый проект
Изменён: 2023.05.17 13:19:29
Создан: 2023.05.17 12:22:48

Новый проект
Изменён: 2023.05.17 12:23:42
Создан: 2023.05.17 12:23:42

Новый проект
Изменён: 2023.05.17 12:21:02
Создан: 2023.05.17 12:21:02

Новый проект
Изменён: 2023.05.17 12:20:06
Создан: 2023.05.17 12:20:06

Новый проект
Изменён: 2023.05.15 23:11:34
Создан: 2023.05.15 23:11:34

Выход

Рисунок 1 - Окно для работы с файлами проектов

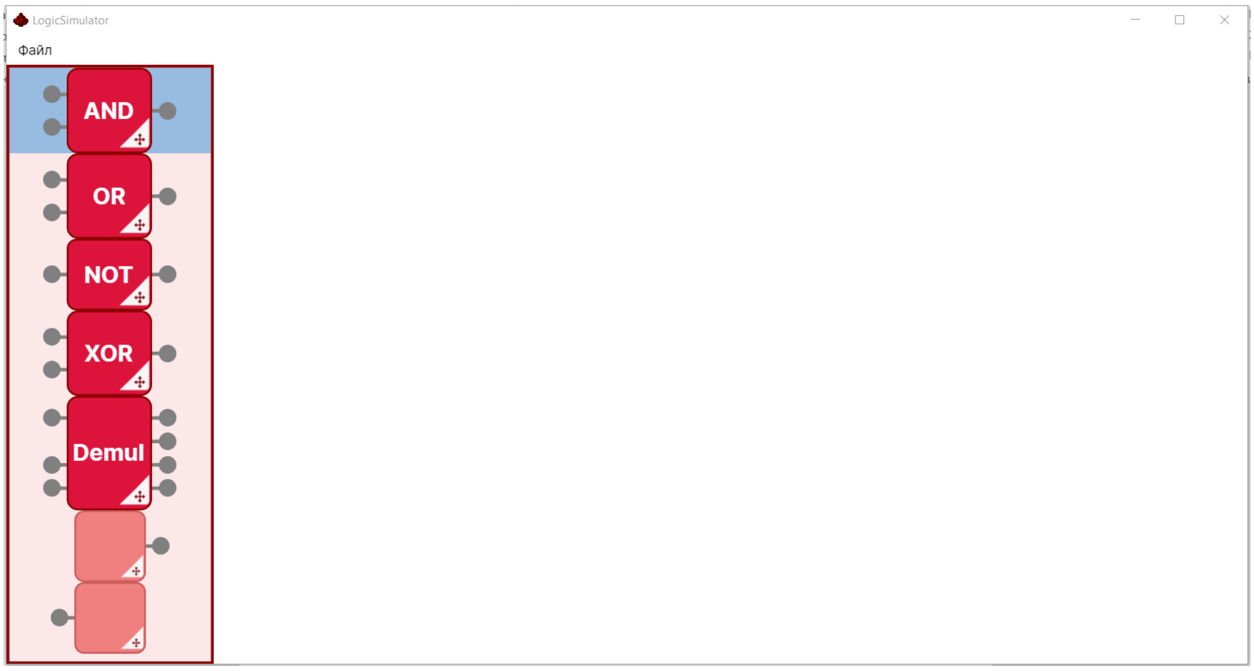


Рисунок 2 – Окно для работы с логическими элементами

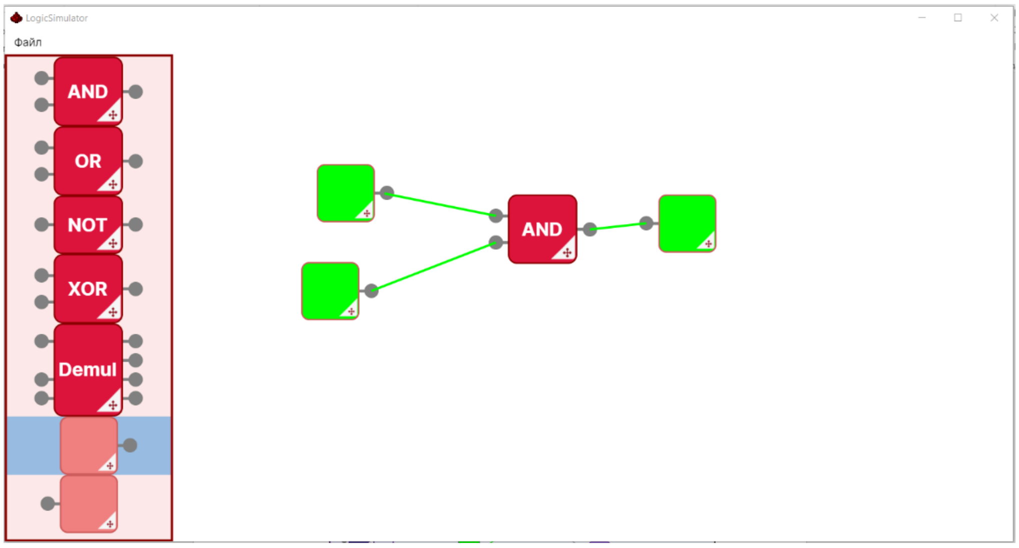


Рисунок 3 – Пример работы логического элемента «И»

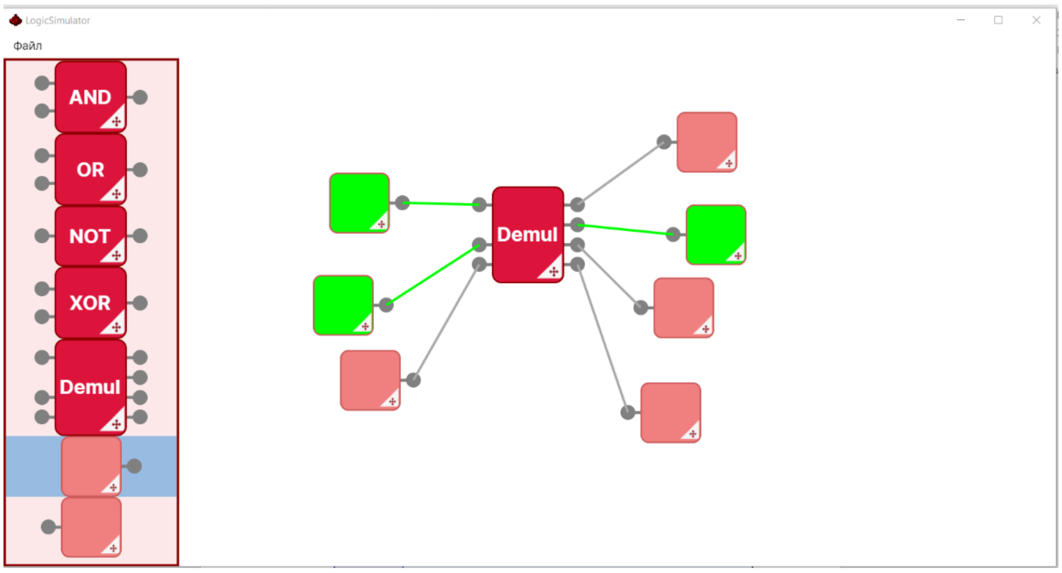


Рисунок 4 – Пример работы демультиплексора

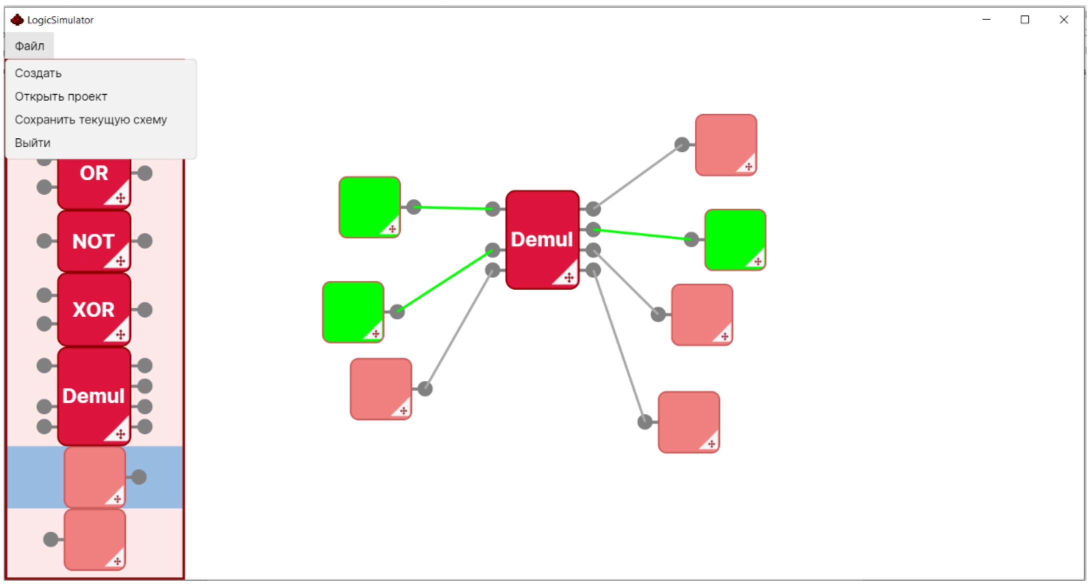


Рисунок 5 – Меню, открывающееся в левой верхней части окна

## 3. Use-case диаграмма

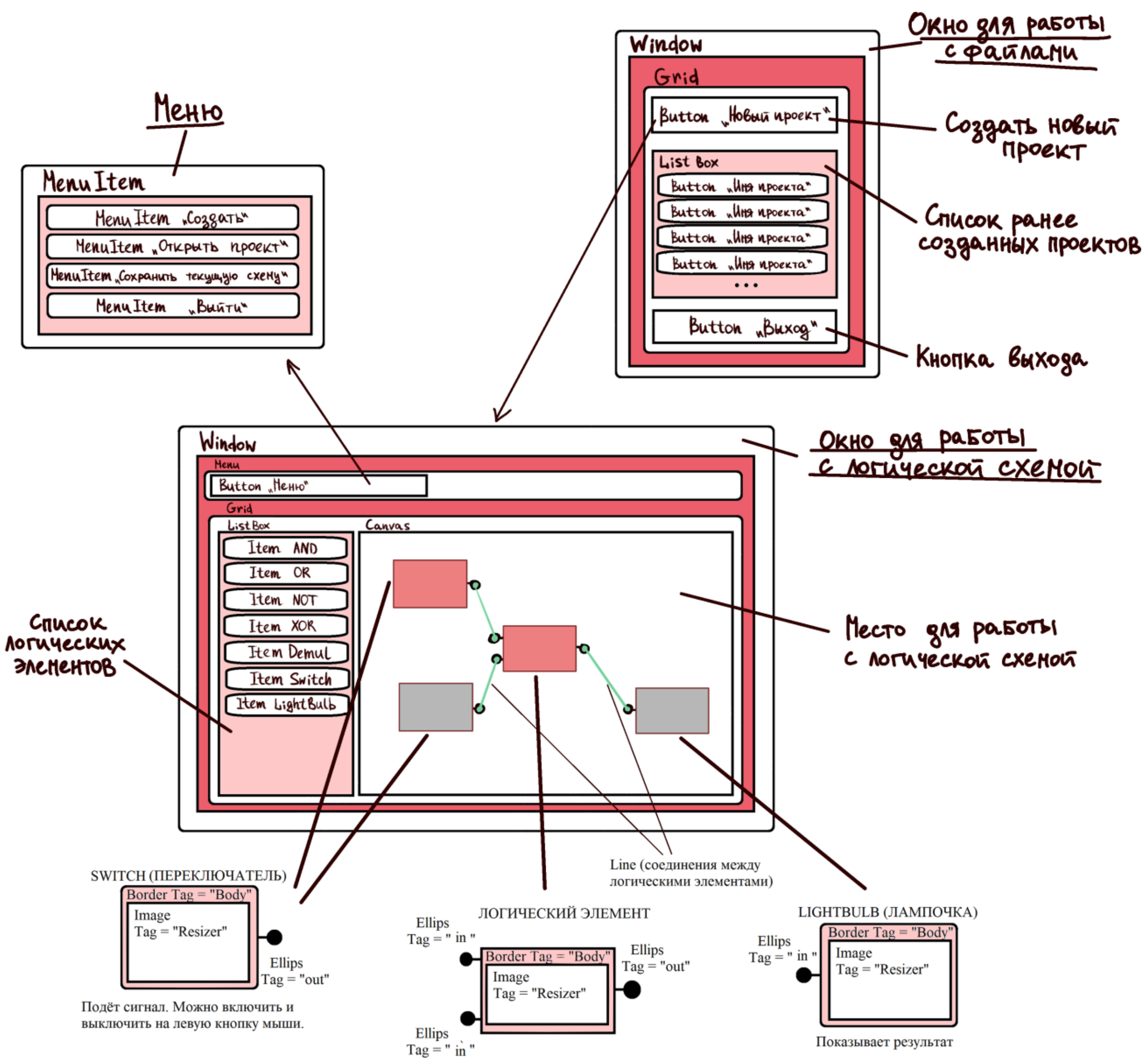


Рисунок 6 - Use-case диаграмма приложения

## 4. Диаграмма классов

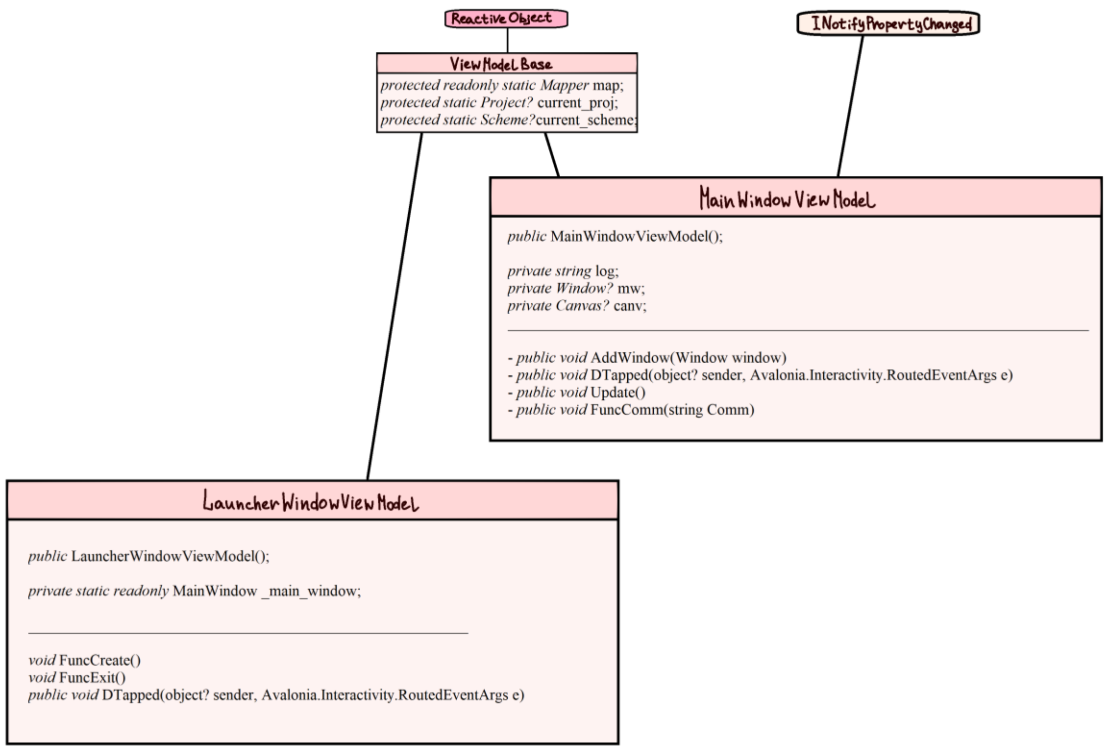

ReactiveObject
INotifyPropertyChanged
ViewModelBase
protected readonly static Mapper map;
protected static Project? current_proj;
protected static Scheme?current_scheme;
MainWindowViewModel
public MainWindowViewModel();
private string log;
private Window? mw;
private Canvas? canv;
- public void AddWindow(Window window)
- public void DTapped(object? sender, Avalonia.Interactivity.RoutedEventArgs e)
- public void Update()
- public void FuncComm(string Comm)
LauncherWindowViewModel
public LauncherWindowViewModel();
private static readonly MainWindow _main_window;
void FuncCreate()
void FuncExit()
public void DTapped(object? sender, Avalonia.Interactivity.RoutedEventArgs e)

Рисунок 7 – классы окон

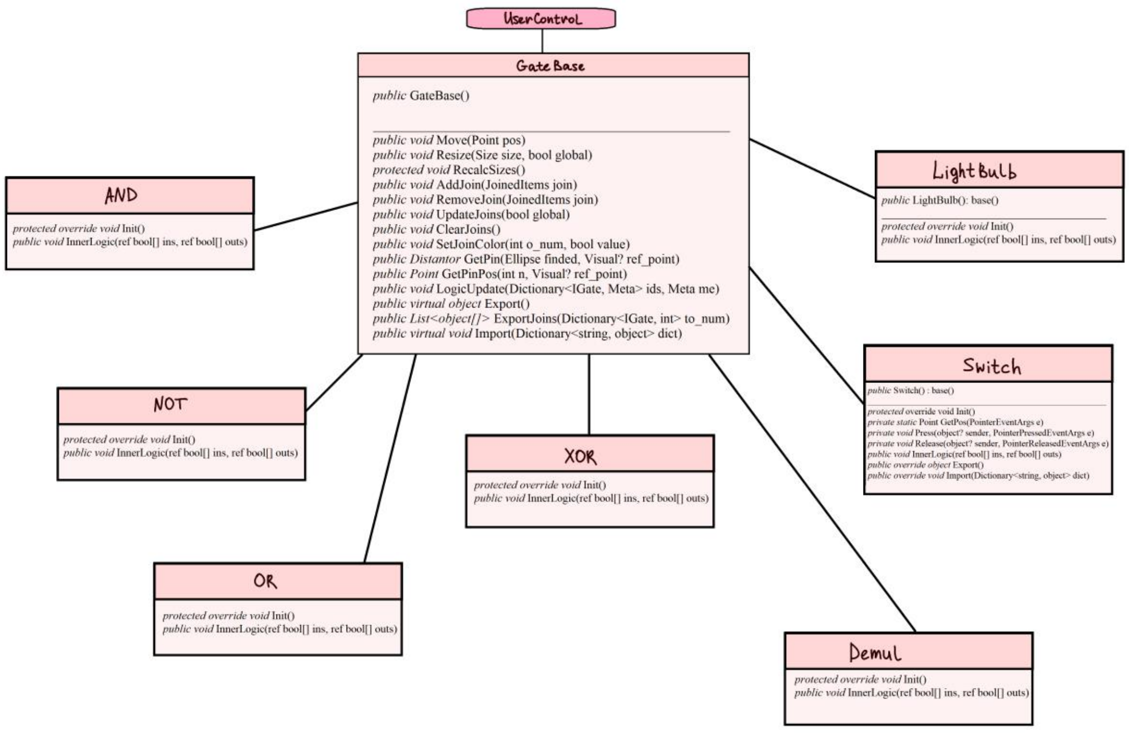
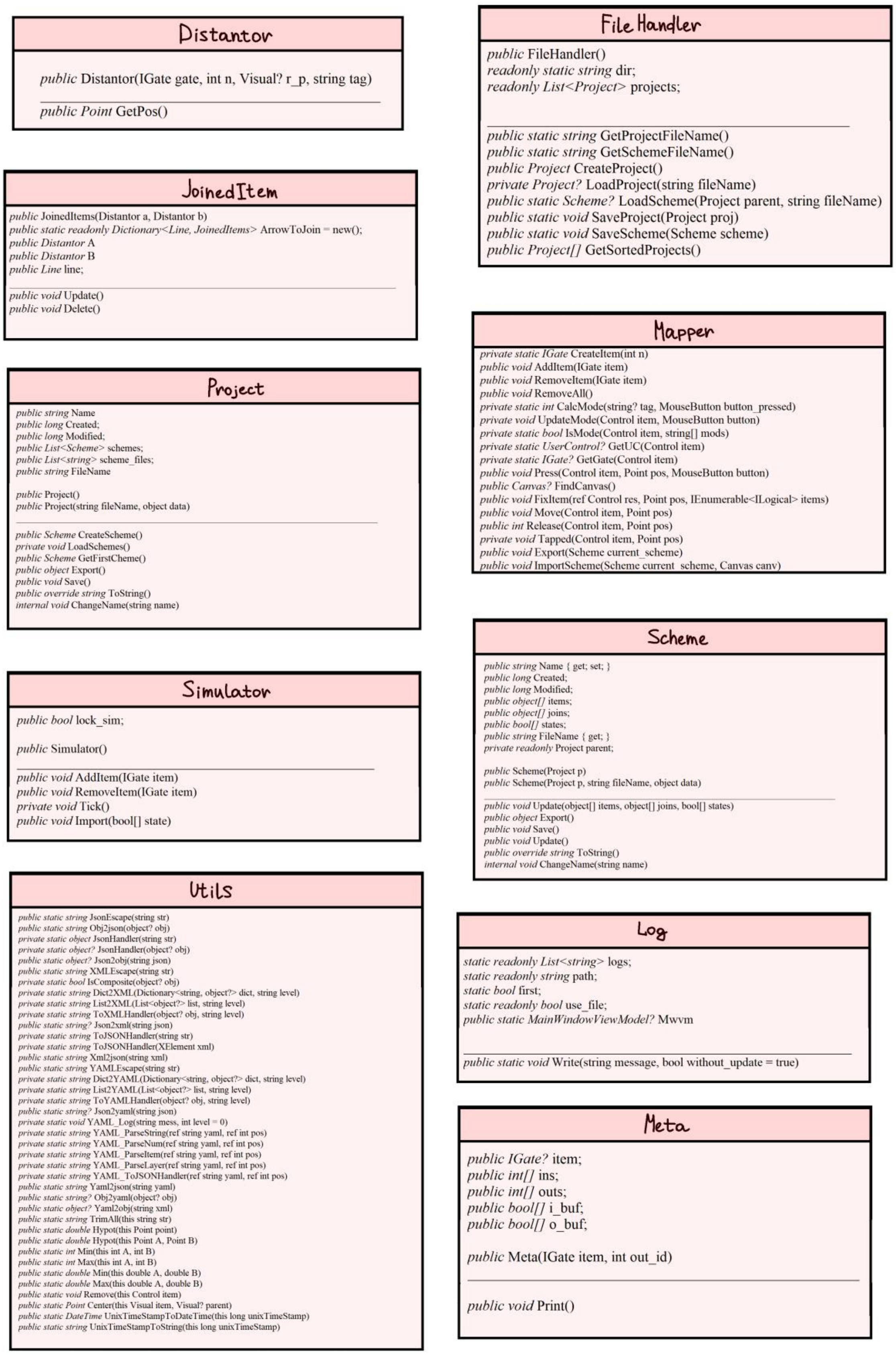

UserControl
GateBase
public GateBase()
public void Move(Point pos)
public void Resize(Size size, bool global)
protected void RecalcSizes()
public void AddJoin(JoinedItems join)
public void RemoveJoin(JoinedItems join)
public void UpdateJoins(bool global)
public void ClearJoins()
public void SetJoinColor(int o_num, bool value)
public Distantor GetPin(Ellipse finded, Visual? ref_point)
public Point GetPinPos(int n, Visual? ref_point)
public void LogicUpdate(Dictionary<IGate, Meta> ids, Meta me)
public virtual object Export()
public List<object[]> ExportJoins(Dictionary<IGate, int> to_num)
public virtual void Import(Dictionary<string, object> dict)
AND
protected override void Init()
public void InnerLogic(ref bool[] ins, ref bool[] outs)
NOT
protected override void Init()
public void InnerLogic(ref bool[] ins, ref bool[] outs)
XOR
protected override void Init()
public void InnerLogic(ref bool[] ins, ref bool[] outs)
OR
protected override void Init()
public void InnerLogic(ref bool[] ins, ref bool[] outs)
LightBulb
public LightBulb(): base()
protected override void Init()
public void InnerLogic(ref bool[] ins, ref bool[] outs)
Switch
public Switch() : base()
protected override void Init()
private static Point GetPos(PointerEventArgs e)
private void Press(object? sender, PointerPressedEventArgs e)
private void Release(object? sender, PointerReleasedEventArgs e)
public void InnerLogic(ref bool[] ins, ref bool[] outs)
public override object Export()
public override void Import(Dictionary<string, object> dict)
Demul
protected override void Init()
public void InnerLogic(ref bool[] ins, ref bool[] outs)

Рисунок 8 – классы логических функций



Distantor

```
public Distantor(IGate gate, int n, Visual? r_p, string tag)

public Point GetPos()
```

JoinedItem

```
public JoinedItems(Distantor a, Distantor b)
public static readonly Dictionary<Line, JoinedItems> ArrowToJoin = new();
public Distantor A
public Distantor B
public Line line;

public void Update()
public void Delete()
```

Project

```
public string Name
public long Created;
public long Modified;
public List<Scheme> schemes;
public List<string> scheme_files;
public string FileName

public Project()
public Project(string fileName, object data)

public Scheme CreateScheme()
private void LoadSchemes()
public Scheme GetFirstCheme()
public object Export()
public void Save()
public override string ToString()
internal void ChangeName(string name)
```

Simulator

```
public bool lock_sim;

public Simulator()

public void AddItem(IGate item)
public void RemoveItem(IGate item)
private void Tick()
public void Import(bool[] state)
```

Utils

```
public static string JsonEscape(string str)
public static string Obj2json(object? obj)
private static object JsonHandler(string str)
private static object? JsonHandler(object? obj)
public static object? Json2obj(string json)
public static string XMLEscape(string str)
private static bool IsComposite(object? obj)
private static string Dict2XML(Dictionary<string, object?> dict, string level)
private static string List2XML(List<object?> list, string level)
private static string ToXMLHandler(object? obj, string level)
public static string? Json2xml(string json)
private static string ToJSONHandler(string str)
private static string ToJSONHandler(XElement xml)
public static string Xml2json(string xml)
public static string YAMLEscape(string str)
private static string Dict2YAML(Dictionary<string, object?> dict, string level)
private static string List2YAML(List<object?> list, string level)
private static string ToYAMLHandler(object? obj, string level)
public static string? Json2yaml(string json)
private static void YAML_Log(string mess, int level = 0)
private static string YAML_ParseString(ref string yaml, ref int pos)
private static string YAML_ParseNum(ref string yaml, ref int pos)
private static string YAML_ParseItem(ref string yaml, ref int pos)
private static string YAML_ParseLayer(ref string yaml, ref int pos)
private static string YAML_ToJSONHandler(ref string yaml, ref int pos)
public static string Yaml2json(string yaml)
public static string? Obj2yaml(object? obj)
public static object? Yaml2obj(string xml)
public static string TrimAll(this string str)
public static double Hypot(this Point point)
public static double Hypot(this Point A, Point B)
public static int Min(this int A, int B)
public static int Max(this int A, int B)
public static double Min(this double A, double B)
public static double Max(this double A, double B)
public static void Remove(this Control item)
public static Point Center(this Visual item, Visual? parent)
public static DateTime UnixTimeStampToDateTime(this long unixTimeStamp)
public static string UnixTimeStampToString(this long unixTimeStamp)
```

File Handler

```
public FileHandler()
readonly static string dir;
readonly List<Project> projects;

public static string GetProjectFileName()
public static string GetSchemeFileName()
public Project CreateProject()
private Project? LoadProject(string fileName)
public static Scheme? LoadScheme(Project parent, string fileName)
public static void SaveProject(Project proj)
public static void SaveScheme(Scheme scheme)
public Project[] GetSortedProjects()
```

Mapper

```
private static IGate CreateItem(int n)
public void AddItem(IGate item)
public void RemoveItem(IGate item)
public void RemoveAll()
private static int CalcMode(string? tag, MouseButton button_pressed)
private void UpdateMode(Control item, MouseButton button)
private static bool IsMode(Control item, string[] mods)
private static UserControl? GetUC(Control item)
private static IGate? GetGate(Control item)
public void Press(Control item, Point pos, MouseButton button)
public Canvas? FindCanvas()
public void FixItem(ref Control res, Point pos, IEnumerable<ILogical> items)
public void Move(Control item, Point pos)
public int Release(Control item, Point pos)
private void Tapped(Control item, Point pos)
public void Export(Scheme current_scheme)
public void ImportScheme(Scheme current_scheme, Canvas canv)
```

Scheme

```
public string Name { get; set; }
public long Created;
public long Modified;
public object[] items;
public object[] joins;
public bool[] states;
public string FileName { get; }
private readonly Project parent;

public Scheme(Project p)
public Scheme(Project p, string fileName, object data)

public void Update(object[] items, object[] joins, bool[] states)
public object Export()
public void Save()
public void Update()
public override string ToString()
internal void ChangeName(string name)
```

Log

```
static readonly List<string> logs;
static readonly string path;
static bool first;
static readonly bool use_file;
public static MainWindowViewModel? Mwvm

public static void Write(string message, bool without_update = true)
```

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)

public void Print()
```

Рисунок 9 – вспомогательные классы

## 5. Заключение

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

# 6. Листинг программы

## Models/destinator.cs

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

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

        readonly Visual? _ref_point;

        public Distantor(IGate gate, int n, Visual? r_p, string tag)
        {
            this.parent = gate;
            num = n; // Например, в AND_2-gate: 0 и 1 - входы, 2 - выход
            _ref_point = r_p;
            this.tag = tag;
        }

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

## Models/mapper.cs

```
using Avalonia.Controls;
using Avalonia;
using LogicSimulator.ViewModels;
using LogicSimulator.Views.Shapes;
using System;
using System.Collections.Generic;
using DynamicData;
using Avalonia.Controls.Shapes;
using Avalonia.Media;
using Avalonia.LogicalTree;
using System.Linq;
using System.Threading.Tasks;
using Avalonia.Threading;
using Avalonia.Input;

namespace LogicSimulator.Models
{
    public class Mapper
    {
        readonly Line marker = new()
        {
            Tag = "Marker",
            ZIndex = 2,
            IsVisible = false,
            Stroke = Brushes.YellowGreen,
            StrokeThickness = 3
        };
        public Line Marker { get => marker; }

        readonly Simulator sim = new();
```

```
public int SelectedItem { get; set; }

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 Demul(),
      5 => new Switch(),
      6 => new LightBulb(),
      _ => throw new ArgumentOutOfRangeException(),
   };
}

public IGate[] item_types = new IGate[] {
   CreateItem(0),
   CreateItem(1),
   CreateItem(2),
   CreateItem(3),
   CreateItem(4),
   CreateItem(5),
   CreateItem(6),
};

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



readonly List<IGate> items = new();
public void AddItem(IGate item)
{
   items.Add(item);
   sim.AddItem(item);
}
public void RemoveItem(IGate item)
{
   items.Remove(item);
   sim.RemoveItem(item);

   item.ClearJoins();
   ((Control)item).Remove();
}
public void RemoveAll()
{
   foreach (var item in items.ToArray()) RemoveItem(item);
}


int mode = 0;
/*
 *   Режимы перемещения:
 * 0 - ничего не делает
 * 1 - двигаем камеру
 * 2 - двигаем элемент
 * 3 - тянем элемент
 * 4 - удаляем 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, MouseButton button_pressed)
```

```
{
   if (tag == null) return 0;
   if (button_pressed == MouseButton.Right) return 4;

   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, MouseButton button) => mode = CalcMode((string?)item.Tag, button);

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) => GetUC(item) as IGate;

/*
 * Обработка мыши
 */

Point moved_pos;
IGate? moved_item;
Point item_old_pos;
Size item_old_size;

Ellipse? marker_circle;
Distantor? start_dist;
int marker_mode;

Line? old_join;
bool join_start;

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

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

   switch (mode)
   {
      case 3:
         if (moved_item == null) break;
```

```
                item_old_size = moved_item.GetBodySize();
                break;
            case 4:
                if (item is not Line @join1) break;
                old_join = @join1;
                break;
            case 5 or 6 or 7:
                if (marker_circle == null) break;
                var gate = GetGate(marker_circle) ?? throw new Exception();
                start_dist = gate.GetPin(marker_circle, FindCanvas());

                var circle_pos = start_dist.GetPos();
                marker.StartPoint = marker.EndPoint = circle_pos;
                marker.IsVisible = true;
                marker_mode = mode;
                break;
            case 8:
                if (item is not Line @join) break;
                JoinedItems.ArrowToJoin.TryGetValue(@join, out var @join2);
                if (@join2 == null) break;

                var dist_a = @join.StartPoint.Hypot(pos);
                var dist_b = @join.EndPoint.Hypot(pos);
                join_start = dist_a > dist_b;
                old_join = @join;

                marker.StartPoint = join_start ? @join.StartPoint : pos;
                marker.EndPoint = join_start ? pos : @join.EndPoint;
                marker_mode = CalcMode(join_start ? @join2.A.tag : @join2.B.tag, button);

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

        Move(item, pos);
    }

    public Canvas? FindCanvas()
    {
        foreach (var item in items)
        {
            var p = item.GetSelf().Parent;
            if (p is Canvas @canv) return @canv;
        }
        return null;
    }
    public void FixItem(ref Control res, Point pos, IEnumerable<ILogical> items)
    {
        foreach (var logic in items)
        {
            var item = (Control)logic;
            var tb = item.TransformedBounds;
            if (tb != null && tb.Value.Bounds.TransformToAABB(tb.Value.Transform).Contains(pos) &&
(string?)item.Tag != "Join") res = item;
            FixItem(ref res, pos, item.GetLogicalChildren());
        }
    }
    public void Move(Control item, Point pos)
    {
        if (mode == 5 || mode == 6 || mode == 7 || mode == 8)
        {
            var canv = FindCanvas();
            if (canv != null)
            {
                var tb = canv.TransformedBounds;
            if (tb != null)
            {
                item = new Canvas() { Tag = "Scene" };
                var bounds = tb.Value.Bounds.TransformToAABB(tb.Value.Transform);
                FixItem(ref item, pos + bounds.TopLeft, canv.Children);
            }
        }
    }

    string[] mods = new[] { "In", "Out", "IO" };
    var tag = (string?)item.Tag;
    if (IsMode(item, mods) && item is Ellipse @ellipse
        && !(marker_mode == 5 && tag == "In" || marker_mode == 6 && tag == "Out"))
    {
        if (marker_circle != null && marker_circle != @ellipse)
        {
            marker_circle.Fill = new SolidColorBrush(Color.Parse("Gray"));
            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("Gray"));
        marker_circle.Stroke = Brushes.Gray;
        marker_circle = null;
    }


    var delta = pos - moved_pos;
    if (delta.X == 0 && delta.Y == 0) return;

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

    switch (mode)
    {
        case 2:
            if (moved_item == null) break;
            var new_pos = item_old_pos + delta;
            moved_item.Move(new_pos);
            break;
        case 3:
            if (moved_item == null) break;
            var new_size = item_old_size + new Size(delta.X, delta.Y);
            moved_item.Resize(new_size, false);
            break;
        case 5 or 6 or 7:
            var end_pos = marker_circle == null ? pos : marker_circle.Center(FindCanvas());
            marker.EndPoint = end_pos;
            break;
        case 8:
            if (old_join == null) break;
            var p = marker_circle == null ? pos : marker_circle.Center(FindCanvas());
            if (join_start) marker.EndPoint = p;
            else marker.StartPoint = p;
            break;
    }
}

// Обрабатывается после Release
public bool tapped = false;
public Point tap_pos;
public Line? new_join;
public int Release(Control item, Point pos)
{
   Move(item, pos);

   switch (mode)
   {
      case 4:

         if (old_join == null) break;
         JoinedItems.ArrowToJoin.TryGetValue(old_join, out var @join);
         /*
         if (marker_circle != null && @join != null)
         {
            IGate? gate = GetGate(marker_circle) ?? throw new Exception();
            Distantor? p = gate.GetPin(marker_circle, FindCanvas());
            @join.Delete();

            var newy = join_start ? new JoinedItems(@join.A, p) : new JoinedItems(p, @join.B);
            new_join = newy.line;
         }
         else old_join.IsVisible = true;

         marker.IsVisible = false;
         marker_mode = 0;
         old_join = null;
         */
         //Удаление соединяющей линии
         @join?.Delete();
         break;

      case 5 or 6 or 7:
         if (start_dist == null) break;
         if (marker_circle != null)
         {
            var gate = GetGate(marker_circle) ?? throw new Exception();
            var end_dist = gate.GetPin(marker_circle, FindCanvas());

            if (start_dist.parent.GetSelf() != end_dist.parent.GetSelf())
            {
               var newy = new JoinedItems(start_dist, end_dist);
               new_join = newy.line;
            }
         }
         marker.IsVisible = false;
         marker_mode = 0;
         break;
      case 8:
         if (old_join == null) break;
         JoinedItems.ArrowToJoin.TryGetValue(old_join, out var @join1);
         if (marker_circle != null && @join1 != null)
         {
            IGate? gate = GetGate(marker_circle) ?? throw new Exception();
            Distantor? p = gate.GetPin(marker_circle, FindCanvas());
            @join1.Delete();

            var newy = join_start ? new JoinedItems(@join1.A, p) : new JoinedItems(p, @join1.B);
            new_join = newy.line;
         }
         else old_join.IsVisible = true;

         marker.IsVisible = false;
         marker_mode = 0;
         old_join = null;

         //Удаление соединяющей линии
         @join1?.Delete();
```

```
            break;


    }

    if (tapped) Tapped(item, pos);

    int res_mode = mode;
    mode = 0;
    return res_mode;
}

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

    if (mode == 4 && moved_item != null)
    {
        RemoveItem(moved_item);
    }
}



public readonly FileHandler filer = new();

public void Export(Scheme current_scheme)
{
    var arr = items.Select(x => x.Export()).ToArray();

    Dictionary<IGate, int> item_to_num = new();
    int n = 0;
    foreach (var item in items) item_to_num.Add(item, n++);
    List<object[]> joins = new();
    foreach (var item in items) joins.Add(item.ExportJoins(item_to_num));

    bool[] states = sim.Export();

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

    Log.Write("Items: " + Utils.Obj2json(arr));
    Log.Write("Joins: " + Utils.Obj2json(joins));
    Log.Write("States: " + Utils.Obj2json(states));
}

public void ImportScheme(Scheme current_scheme, Canvas canv)
{
    sim.lock_sim = true;

    RemoveAll();

    List<IGate> list = new();
    foreach (var item in current_scheme.items)
    {
        if (item is not Dictionary<string, object> @dict) { Log.Write("Не верный тип элемента: " + item); continue; }

        if (!@dict.TryGetValue("id", out var @value)) { Log.Write("id элемента не обнаружен"); continue; }
        if (@value is not int @id) { Log.Write("Неверный тип id: " + @value); continue; }
        var newy = CreateItem(@id);

        newy.Import(@dict);
        AddItem(newy);
        canv.Children.Add(newy.GetSelf());
        list.Add(newy);
    }
```

```
            var items_arr = list.ToArray();

            List<JoinedItems> joinz = new();
            foreach (var obj in current_scheme.joins)
            {
                if (obj is not List<object> @join) { Log.Write("Одно из соединений не того типа: " + obj); continue; }
                if (@join.Count != 6 ||
                    @join[0] is not int @num_a || @join[1] is not int @pin_a || @join[2] is not string @tag_a ||
                    @join[3] is not int @num_b || @join[4] is not int @pin_b || @join[5] is not string @tag_b) {
Log.Write("Содержимое списка соединения ошибочно"); continue; }

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

            sim.Import(current_scheme.states);
            sim.lock_sim = false;

            Task.Run(async () =>
            {
                await Task.Delay(50);
                await Dispatcher.UIThread.InvokeAsync(() =>
                {
                    foreach (var join in joinz) join.Update();
                });
            });
        }
    }
}
```

## Models/project.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace LogicSimulator.Models
{
    public class Project : IComparable //класс проекта, который хранит информацию о нём (дата создания,
редактирования, имя, список схем)
    {
        public string Name { get; private set; }
        public long Created;
        public long Modified;

        public List<Scheme> schemes = new();
        public List<string> scheme_files = new();
        public string FileName { get; }

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

        public Project(string fileName, object data)
        {
            FileName = fileName;

            if (data is not Dictionary<string, object> dict) throw new Exception("Ожидался словарь в корне проекта");
    if (!dict.TryGetValue("name", out var value)) throw new Exception("В проекте нет имени");
    if (value is not string name) throw new Exception("Тип имени проекта - не строка");
    Name = name;

    if (!dict.TryGetValue("created", out var value2)) throw new Exception("В проекте нет времени создания");
    if (value2 is not int create_t) throw new Exception("Время создания проекта - не строка");
    Created = create_t;

    if (!dict.TryGetValue("modified", out var value3)) throw new Exception("В проекте нет времени изменения");
    if (value3 is not int mod_t) throw new Exception("Время изменения проекта - не строка");
    Modified = mod_t;

    if (!dict.TryGetValue("schemes", out var value4)) throw new Exception("В проекте нет списка схем");
    if (value4 is not List<object> arr) throw new Exception("Списко схем проекта - не массив строк");
    foreach (var file in arr)
    {
        if (file is not string str) throw new Exception("Одно из файловых имёт списка схем проекта - не строка");
        scheme_files.Add(str);
    }
}



public Scheme CreateScheme()
{
    var scheme = new Scheme(this);
    schemes.Add(scheme);
    scheme.Save();
    scheme_files.Add(scheme.FileName);
    Save();
    return scheme;
}

bool loaded = false;
private void LoadSchemes()
{
    if (loaded) return;
    foreach (var fileName in scheme_files)
    {
        var scheme = FileHandler.LoadScheme(this, fileName);
        if (scheme != null) schemes.Add(scheme);
    }
    loaded = true;
}
public Scheme GetFirstCheme()
{
    LoadSchemes();
    return schemes[0];
}


public object Export()
{
    return new Dictionary<string, object>
    {
        ["name"] = Name,
        ["created"] = Created,
        ["modified"] = Modified,
        ["schemes"] = schemes.Select(x => x.FileName).ToArray(),
    };
}

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

public int CompareTo(object? obj)
{
```

```
            if (obj is null) throw new ArgumentNullException(nameof(obj));
            if (obj is not Project proj) throw new ArgumentException(nameof(obj));
            return (int)(proj.Modified - Modified);
        }

        public override string ToString()
        {
            return Name + "\nИзменён: " + Modified.UnixTimeStampToString() + "\nСоздан: " +
Created.UnixTimeStampToString();
        }

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

## Models/simulator.cs

```
using LogicSimulator.ViewModels;
using LogicSimulator.Views.Shapes;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace LogicSimulator.Models
{
    public class Meta
    {
        public IGate? item;
        public int[] ins;
        public int[] outs;
        public bool[] i_buf;
        public bool[] o_buf;

        public Meta(IGate item, int out_id)
        {
            this.item = item;
            ins = Enumerable.Repeat(0, item.InputCount).ToArray();
            outs = Enumerable.Range(out_id, item.OutputCount).ToArray();
            i_buf = Enumerable.Repeat(false, item.InputCount).ToArray();
            o_buf = Enumerable.Repeat(false, item.OutputCount).ToArray();
        }

        public void Print()
        {
            Log.Write("Элемент: " + item + " | Ins: " + Utils.Obj2json(ins) + " | Outs: " + Utils.Obj2json(outs));
        }
    }


    public class Simulator
    {
        public bool lock_sim = false;
        public Simulator()
        {
            var task = Task.Run(async () =>
            {
                for (; ; )
                {
                    await Task.Delay(1000 / 60);
            if (lock_sim) continue;
            try { Tick(); }
            catch (Exception e) { Log.Write("Logical crush: " + e); continue; }
        }
    });
}



List<bool> outs = new() { false };
List<bool> outs2 = new() { false };
readonly List<Meta> items = new();
readonly Dictionary<IGate, Meta> ids = new();

public void AddItem(IGate item)
{
    lock_sim = true;

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

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

    lock_sim = false;
}

public void RemoveItem(IGate item)
{
    lock_sim = true;

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

    lock_sim = false;
}

private void Tick()
{
    foreach (var meta in items)
    {
        var item = meta.item;
        if (item == null) continue;

        item.LogicUpdate(ids, meta);

        int[] i_n = meta.ins, o_n = meta.outs;
        bool[] ib = meta.i_buf, ob = meta.o_buf;

        for (int i = 0; i < ib.Length; i++) ib[i] = outs[i_n[i]];
        item.InnerLogic(ref ib, ref ob);
        for (int i = 0; i < ob.Length; i++)
        {
            bool res = ob[i];
            outs2[o_n[i]] = res;
            item.SetJoinColor(i, res);
        }
    }

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

```
        }

        public bool[] Export() => outs.ToArray();
        public void Import(bool[] state)
        {
            if (state.Length == 0) state = new bool[] { false };
            outs = state.ToList();
            outs2 = Enumerable.Repeat(false, state.Length).ToList();
        }
    }
}
```

## Models/FileHandler.cs

```
using LogicSimulator.ViewModels;
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;

namespace LogicSimulator.Models
{
    public class FileHandler
    {
        readonly static string dir = "../../../../storage/";
        readonly List<Project> projects = new();

        public FileHandler()
        {
            if (!Directory.Exists(dir)) Directory.CreateDirectory(dir);
            foreach (var fullname in Directory.EnumerateFiles(dir))
            {
                string name = fullname.Split("/")[^1];
                if (name.StartsWith("proj_")) LoadProject(name);
            }
        }



        public static string GetProjectFileName()
        {
            for (int i = 1; ; i++)
            {
                string name = "proj_" + i + ".json";
                if (!File.Exists(dir + name)) return name;
            }
        }
        public static string GetSchemeFileName()
        {
            for (int i = 1; ; i++)
            {
                string name = "scheme_" + i + ".yaml";
                if (!File.Exists(dir + name)) return name;
            }
        }



        public Project CreateProject()
        {
            var proj = new Project();
            projects.Add(proj);
            return proj;
        }
        private Project? LoadProject(string fileName)
        {
            try
            {
                var obj = Utils.Json2obj(File.ReadAllText(dir + fileName)) ?? throw new DataException("Неверная структура
JSON-файла проекта!");
                var proj = new Project(fileName, obj);
                projects.Add(proj);
                return proj;
            }
            catch (Exception e) { Log.Write("Не удалось загрузить проект:" + Environment.NewLine + e); }
            return null;
        }
        public static Scheme? LoadScheme(Project parent, string fileName)
        {
            try
            {
                var obj = Utils.Yaml2obj(File.ReadAllText(dir + fileName)) ?? throw new DataException("Неверная
структура схемы YAML-файла.");
                var scheme = new Scheme(parent, fileName, obj);
                return scheme;
            }
            catch (Exception e) { Log.Write("Не удалось загрузить схему:" + Environment.NewLine + e); }
            return null;
        }



        public static void SaveProject(Project proj)
        {
            var data = Utils.Obj2json(proj.Export());
            File.WriteAllText(dir + proj.FileName, data);
        }
        public static void SaveScheme(Scheme scheme)
        {
            var data = Utils.Obj2yaml(scheme.Export());
            File.WriteAllText(dir + scheme.FileName, data);
        }

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

## Models/Scheme.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace LogicSimulator.Models
{
    public class Scheme //информация о схеме, хранящая в себе все логические элементы, их связи и состояние
    {
        public string Name { get; set; }
        public long Created;
        public long Modified;

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

        public string FileName { get; }
        private readonly Project parent;

        public Scheme(Project p)
```

```
{
   Created = Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
   Name = "Newy";
   items = joins = Array.Empty<object>();
   states = Array.Empty<bool>();
   FileName = FileHandler.GetSchemeFileName();
   parent = p;
}

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

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

   if (!dict.TryGetValue("name", out var value)) throw new Exception("В схеме нет имени");
   if (value is not string name) throw new Exception("Тип имени схемы - не строка");
   Name = name;

   if (!dict.TryGetValue("created", out var value2)) throw new Exception("В схеме нет времени создания");
   if (value2 is not int create_t) throw new Exception("Время создания схемы - не строка");
   Created = create_t;

   if (!dict.TryGetValue("modified", out var value3)) throw new Exception("В схеме нет времени изменения");
   if (value3 is not int mod_t) throw new Exception("Время изменения схемы - не строка");
   Modified = mod_t;

   if (!dict.TryGetValue("items", out var value4)) throw new Exception("В схеме нет списка элементов");
   if (value4 is not List<object> arr) throw new Exception("Список элементов схемы - не массив объектов");
   items = arr.ToArray();

   if (!dict.TryGetValue("joins", out var value5)) throw new Exception("В схеме нет списка соединений");
   if (value5 is not List<object> arr2) throw new Exception("Список соединений схемы - не массив объектов");
   joins = arr2.ToArray();

   if (!dict.TryGetValue("states", out var value6)) throw new Exception("В схеме нет списка состояний");
   if (value6 is not List<object> arr3) throw new Exception("Список состояний схемы - не массив bool");
   states = arr3.Select(x => (bool)x).ToArray();
}

public void Update(object[] items, object[] joins, bool[] states)
{
   this.items = items;
   this.joins = joins;
   this.states = states;
   Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
   Update();
}



public object Export()
{
   return new Dictionary<string, object>
   {
      ["name"] = Name,
      ["created"] = Created,
      ["modified"] = Modified,
      ["items"] = items,
      ["joins"] = joins,
      ["states"] = states,
   };
}
public void Save() => FileHandler.SaveScheme(this);
public void Update()
```

```
        {
            Modified = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
            parent.Modified = Modified;
            parent.Save();
            Save();
        }

        public override string ToString() => Name;

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

## Models/JoinedItems.cs

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

namespace LogicSimulator.Models
{
    public class JoinedItems //соединение между логическими элементами
    {
        public static readonly Dictionary<Line, JoinedItems> ArrowToJoin = new();

        public JoinedItems(Distantor a, Distantor b)
        {
            A = a;
            B = b;
            Update();

            a.parent.AddJoin(this);
            b.parent.AddJoin(this);
            ArrowToJoin[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()
        {
            ArrowToJoin.Remove(line);
            line.Remove();
            A.parent.RemoveJoin(this);
            B.parent.RemoveJoin(this);
        }
    }
}
```

## ViewModels/LauncherWindowViewModel.cs

```
using Avalonia.Controls.Presenters;
using Avalonia.Controls;
using ReactiveUI;
using System.Reactive;
using LogicSimulator.Views;
using LogicSimulator.Models;

namespace LogicSimulator.ViewModels
{
    public class LauncherWindowViewModel : ViewModelBase
    {
        Window? me;
        private static readonly MainWindow _main_window = new();

        public LauncherWindowViewModel()
        {
            Create = ReactiveCommand.Create<Unit, Unit>(_ => { FuncCreate(); return new Unit(); });
            Exit = ReactiveCommand.Create<Unit, Unit>(_ => { FuncExit(); return new Unit(); });
        }
        public void AddWindow(Window window) => me = window;

        void FuncCreate()
        {
            var newy = map.filer.CreateProject();
            current_proj = newy;
            current_scheme = current_proj.GetFirstCheme();
            _main_window.Show();
            _main_window.Update();
            me?.Close();
        }
        void FuncExit()
        {
            me?.Close();
        }

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


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

        public void DTapped(object? sender, Avalonia.Interactivity.RoutedEventArgs e)
        {
            Control? src = (Control?)e.Source;

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

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

            current_proj = proj;
            current_scheme = current_proj.GetFirstCheme();
            _main_window.Show();
            _main_window.Update();
            me?.Close();
        }
    }
}
```

## ViewModels/MainWindowViewModel.cs

```
using Avalonia;
using Avalonia.Controls;
using Avalonia.Controls.Presenters;
using Avalonia.Input;
using LogicSimulator.Models;
using LogicSimulator.Views;
using LogicSimulator.Views.Shapes;
using ReactiveUI;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Reactive;

namespace LogicSimulator.ViewModels {
    public class Log {
        static readonly List<string> logs = new();
        static readonly string path = "../../../Log.txt";
        static bool first = true;

        static readonly bool use_file = false;

        public static MainWindowViewModel? Mwvm { private get; set; }
        public static void Write(string message, bool without_update = true) {
            if (!without_update) {
                foreach (var mess in message.Split(Environment.NewLine)) logs.Add(mess);
                while (logs.Count > 50) logs.RemoveAt(0);

                if (Mwvm != null) Mwvm.Logg = string.Join(Environment.NewLine, logs);
            }

            if (use_file) {
                if (first) File.WriteAllText(path, message + Environment.NewLine);
                else File.AppendAllText(path, message + Environment.NewLine);
                first = false;
            }
        }
    }

    public class MainWindowViewModel: ViewModelBase, INotifyPropertyChanged {
        private string log = "";
        public string Logg { get => log; set {
            if (log == value) return;
            log = value;
            PropertyChanged?.Invoke(this, new(nameof(Logg)));
        } }

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

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

            mw = window;
            this.canv = canv;
            if (canv == null) return;

            canv.Children.Add(map.Marker);

            Panel? panel = (Panel?) canv.Parent;
            if (panel == null) return;


            panel.PointerPressed += (object? sender, PointerPressedEventArgs e) => {
                if (e.Source != null && e.Source is Control @control) map.Press(@control, e.GetCurrentPoint(canv).Position,
e.MouseButton);
            };
            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;

                var newy = map.GenSelectedItem();
                var size = newy.GetSize() / 2;
                newy.Move(pos - new Point(size.Width, size.Height));
                canv.Children.Add(newy.GetSelf());
                map.AddItem(newy);
            }

            if (map.new_join != null) {
                canv.Children.Add(map.new_join);
                map.new_join = null;
            }
        }
    };
}

public static IGate[] ItemTypes { get => map.item_types; }
public static int SelectedItem { get => map.SelectedItem; set => map.SelectedItem = value; }



Border? cur_border;
TextBlock? old_b_child;
object? old_b_child_tag;
string? prev_scheme_name;

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

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



public void DTapped(object? sender, Avalonia.Interactivity.RoutedEventArgs e) {
    var src = (Control?) e.Source;

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

    if (src is not TextBlock tb) return;

    var p = tb.Parent;
    if (p == null || p is not Border b) return;

    if (cur_border != null && old_b_child != null) cur_border.Child = old_b_child;
    cur_border = b;
    old_b_child = tb;
    old_b_child_tag = tb.Tag;
    prev_scheme_name = tb.Text;

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

    b.Child = 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") current_proj?.ChangeName(newy.Text);
            else if (old_b_child_tag is Scheme scheme) scheme.ChangeName(newy.Text);
        }
```

```
            b.Child = tb;
            cur_border = null; old_b_child = null;
        };
    }

#pragma warning disable CS0108
    public event PropertyChangedEventHandler? PropertyChanged;
#pragma warning restore CS0108
    public void Update() {
        Log.Write("Текущий проект:" + Environment.NewLine + current_proj);

        if (current_scheme == null || canv == null) throw new Exception();
        map.ImportScheme(current_scheme, canv);

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



    public void FuncComm(string Comm) {
        Log.Write("Comm: " + Comm);
        switch (Comm) {
        case "Create":
            break;
        case "Open":
            new LauncherWindow().Show();
            mw?.Hide();
            break;
        case "Save":
            if (current_scheme != null) map.Export(current_scheme);
            break;
        case "Exit":
            mw?.Close();
            break;
        }
    }

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

## Models/ViewModelBase.cs

```
using LogicSimulator.Models;
using ReactiveUI;

namespace LogicSimulator.ViewModels {
  public class ViewModelBase: ReactiveObject {
    protected readonly static Mapper map = new();
    protected static Project? current_proj;
    protected static Scheme? current_scheme;
  }
}
```

## Views/Shapes/GateBase.cs

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

```
namespace LogicSimulator.Views.Shapes {
  public abstract class GateBase: UserControl
  {  //Абстрактный класс GateBase, по его шаблону создаются другие логические элементы
    public abstract int InputCount { get; }
    public abstract int OutputCount { get; }
    public abstract UserControl GetSelf(); //UserControl - надо глянуть документацию
    protected abstract IGate GetSelfI { get; }
    protected abstract void Init();

    protected Ellipse[] pins;

    public GateBase() {
      Init();
      int count = InputCount + OutputCount;

      List<Ellipse> list = new();
      foreach (var logic in LogicalChildren[0].LogicalChildren)
        if (logic is Ellipse @ellipse) list.Add(@ellipse);
      if (list.Count != count) throw new Exception();
      pins = list.ToArray();

      joins = new JoinedItems?[count];
    }

    public void Move(Point pos) {
      Margin = new(pos.X, pos.Y, 0, 0);
      UpdateJoins(false);
    }

    public void Resize(Size size, bool global) {
      double limit = (9 + 32) * 2;
      width = size.Width.Max(limit / 3 * (InputCount == 0 || OutputCount == 0 ? 2.25 : 3));
      height = size.Height.Max(limit / 3 * (1.5 + 0.75 * InputCount.Max(OutputCount)));
      RecalcSizes();
      UpdateJoins(global);
    }

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


    protected readonly double base_size = 25;
    protected double width = 30 * 3;
    protected double height = 30 * 3;

    public double BaseSize => base_size;
    public double BaseFraction => base_size / 40;
    public double EllipseSize => BaseFraction * 30;

    public Thickness BodyStrokeSize => new(BaseFraction * 3);
    public double EllipseStrokeSize => BaseFraction * 5;
    public double PinStrokeSize => BaseFraction * 6;

    public Thickness BodyMargin => new(base_size, 0, 0, 0);
    public double BodyWidth => width;
    public double BodyHeight => height;
    public CornerRadius BodyRadius => new(width.Min(height) / 10 + BodyStrokeSize.Top);

    public double UC_Width => base_size * 2 + width;
    public double UC_Height => height;

    public double FontSizze => 24;

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



    public abstract Point[][] PinPoints { get; }
    public Thickness[] EllipseMargins { get {
        Point[][] pins = PinPoints;
        double R2 = EllipseSize / 2;
        double X = UC_Width - EllipseSize;
        int n = 0;
        List<Thickness> list = new();
        foreach (var pin_line in pins)
            list.Add(new(n++ < InputCount ? 0 : X, pin_line[0].Y - R2, 0, 0));
        return list.ToArray();
    } }



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

    protected void RecalcSizes() {
        PropertyChanged?.Invoke(this, new(nameof(EllipseSize)));
        PropertyChanged?.Invoke(this, new(nameof(BodyStrokeSize)));
        PropertyChanged?.Invoke(this, new(nameof(EllipseStrokeSize)));
        PropertyChanged?.Invoke(this, new(nameof(PinStrokeSize)));
        PropertyChanged?.Invoke(this, new(nameof(BodyMargin)));
        PropertyChanged?.Invoke(this, new(nameof(BodyWidth)));
        PropertyChanged?.Invoke(this, new(nameof(BodyHeight)));
        PropertyChanged?.Invoke(this, new(nameof(BodyRadius)));
        PropertyChanged?.Invoke(this, new(nameof(EllipseMargins)));
        PropertyChanged?.Invoke(this, new(nameof(PinPoints)));
        PropertyChanged?.Invoke(this, new(nameof(UC_Width)));
        PropertyChanged?.Invoke(this, new(nameof(UC_Height)));
        PropertyChanged?.Invoke(this, new(nameof(FontSizze)));
        PropertyChanged?.Invoke(this, new(nameof(ImageMargins)));

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

    /*
     * Обработка соединений
     */

    protected JoinedItems?[] joins;

    public void AddJoin(JoinedItems join) {
        if (join.A.parent == this) {
            int n = join.A.num;
            joins[n]?.Delete();
            joins[n] = join;
        }
        if (join.B.parent == this) {
            int n = join.B.num;
            joins[n]?.Delete();
            joins[n] = join;
        }
        skip_upd = false;
```

```
}

public void RemoveJoin(JoinedItems join) {
    if (join.A.parent == this) joins[join.A.num] = null;
    if (join.B.parent == this) joins[join.B.num] = null;
    skip_upd = false;
}

public void UpdateJoins(bool global) {
    foreach (var join in joins)
        if (join != null && (!global || join.A.parent == this)) join.Update();
}

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

public void SetJoinColor(int o_num, bool value) {
    var join = joins[o_num + InputCount];
    if (join != null)
        Dispatcher.UIThread.InvokeAsync(() => {
            join.line.Stroke = value ? Brushes.Lime : Brushes.DarkGray;
        });
}


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

public Point GetPinPos(int n, Visual? ref_point) {
    var pin = pins[n];
    return pin.Center(ref_point);
}


bool skip_upd = true;
public void LogicUpdate(Dictionary<IGate, Meta> ids, Meta me) {
    if (skip_upd) return;
    skip_upd = true;

    int ins = InputCount;
    for (int i = 0; i < ins; i++) {
        var join = joins[i];
        if (join == null) { me.ins[i] = 0; continue; }

        if (join.A.parent == this) {
            var item = join.B;
            if (item.tag == "Out" || item.tag == "IO") {
                var p = item.parent;
                Meta meta = ids[p];
                me.ins[i] = meta.outs[item.num - p.InputCount];
            }
        }
        if (join.B.parent == this) {
            var item = join.A;
            if (item.tag == "Out" || item.tag == "IO") {
                var p = item.parent;
                Meta meta = ids[p];
                me.ins[i] = meta.outs[item.num - p.InputCount];
            }
```

```
                }
            }
        }


        public abstract int TypeId { get; }

        public virtual object Export() {
            return new Dictionary<string, object> {
                ["id"] = TypeId,
                ["pos"] = GetPos(),
                ["size"] = GetBodySize()
            };
        }

        public List<object[]> ExportJoins(Dictionary<IGate, int> to_num) {
            List<object[]> res = new();
            int n = 0, ins = InputCount;
            foreach (var join in joins) {
                if (++n > ins) break;
                if (join == null) continue;
                Distantor a = join.A, b = join.B;
                res.Add(new object[] {
                    to_num[a.parent], a.num, a.tag,
                    to_num[b.parent], b.num, b.tag,
                });
            }
            return res;
        }

        public virtual void Import(Dictionary<string, object> dict) {
            if (!@dict.TryGetValue("pos", out var @value)) { Log.Write("pos-запись элемента не обнаружен"); return; }
            if (@value is not Point @pos) { Log.Write("Неверный тип pos-записи элемента: " + @value); return; }
            Move(@pos);

            if (!@dict.TryGetValue("size", out var @value2)) { Log.Write("size-запись элемента не обнаружен"); return; }
            if (@value2 is not Size @size) { Log.Write("Неверный тип size-записи элемента: " + @value2); return; }
            Resize(@size, false);
        }
    }
}
```

## Views/Shapes/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 InputCount { get; }
        public int OutputCount { get; }
        public UserControl GetSelf();

        public Point GetPos();
        public Size GetSize();
        public Size GetBodySize();
        public void Move(Point pos);
        public void Resize(Size size, bool global);

        public Distantor GetPin(Ellipse finded, Visual? ref_point);
        public Point GetPinPos(int n, Visual? ref_point);

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

        public void InnerLogic(ref bool[] ins, ref bool[] outs);
        public void LogicUpdate(Dictionary<IGate, Meta> ids, Meta me);

        public int TypeId { get; }
        public object Export();
        public List<object[]> ExportJoins(Dictionary<IGate, int> to_num);
        public void Import(Dictionary<string, object> dict);
    }
}
```

## Views/Shapes/AND_2.axaml.cs

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

namespace LogicSimulator.Views.Shapes {
    public partial class AND_2: GateBase, IGate, INotifyPropertyChanged {
        public override int TypeId => 0;

        public override int InputCount => 2;
        public override int OutputCount => 1;
        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;

        protected override void Init() {
            height = 30 * 3;
            InitializeComponent();
            DataContext = this;
        }

        public override Point[][] PinPoints { get {
            double X = EllipseSize - EllipseStrokeSize / 2;
            double X2 = base_size + width - EllipseStrokeSize / 2;
            double R = BodyRadius.TopLeft;
            double Y_s = R, Y_m = height / 2, Y_e = height - Y_s;
            double min = EllipseSize + BaseFraction * 2;
            double Y = Y_s + (Y_e - Y_s) / 4;
            double Y2 = Y_s + (Y_e - Y_s) / 4 * 3;
            if (Y2 - Y < min) { Y = Y_m - min / 2; Y2 = Y_m + min / 2; }
            double PinWidth = base_size - EllipseSize + PinStrokeSize;
            return new Point[][] {
                new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход
                new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход
                new Point[] { new(X2, Y_m), new(X2 + PinWidth, Y_m) }, // Единственный выход
            };
        } }


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

## Views/Shapes/NOT.axaml.cs

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

namespace LogicSimulator.Views.Shapes {
    public partial class NOT: GateBase, IGate, INotifyPropertyChanged {
        public override int TypeId => 2;
        public override int InputCount => 1;
        public override int OutputCount => 1;
        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;

        protected override void Init() {
            height = 30 * 2.5;
            InitializeComponent();
            DataContext = this;
        }


        public override Point[][] PinPoints { get {
            double X = EllipseSize - EllipseStrokeSize / 2;
            double X2 = base_size + width - EllipseStrokeSize / 2;
            double Y = height / 2;
            double PinWidth = base_size - EllipseSize + PinStrokeSize;
            return new Point[][] {
                new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Единственный вход
                new Point[] { new(X2, Y), new(X2 + PinWidth, Y) }, // Единственный выход
            };
        } }


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

## Views/Shapes/OR_2.axaml.cs

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

namespace LogicSimulator.Views.Shapes {
    public partial class OR_2: GateBase, IGate, INotifyPropertyChanged {
        public override int TypeId => 1;

        public override int InputCount => 2;
        public override int OutputCount => 1;
        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;

        protected override void Init() {
            height = 30 * 3;
            InitializeComponent();
            DataContext = this;
        }


        public override Point[][] PinPoints { get {
            double X = EllipseSize - EllipseStrokeSize / 2;
            double X2 = base_size + width - EllipseStrokeSize / 2;
            double R = BodyRadius.TopLeft;
            double Y_s = R, Y_m = height / 2, Y_e = height - Y_s;
            double min = EllipseSize + BaseFraction * 2;

            double Y = Y_s + (Y_e - Y_s) / 4;
            double Y2 = Y_s + (Y_e - Y_s) / 4 * 3;
            if (Y2 - Y < min) { Y = Y_m - min / 2; Y2 = Y_m + min / 2; }
            double PinWidth = base_size - EllipseSize + PinStrokeSize;
            return new Point[][] {
                new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход
                new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход
                new Point[] { new(X2, Y_m), new(X2 + PinWidth, Y_m) }, // Единственный выход
            };
        } }



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

## Views/Shapes/Switch.axaml.cs

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

namespace LogicSimulator.Views.Shapes {
    public partial class Switch: GateBase, IGate, INotifyPropertyChanged {
        public override int TypeId => 5;

        public override int InputCount => 0;
        public override int OutputCount => 1;
        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;

        protected override void Init() {
            width = 30 * 2.5;
            height = 30 * 2.5;
            InitializeComponent();
            DataContext = this;
        }

        readonly Border border;
        public Switch() : base() {
            if (LogicalChildren[0].LogicalChildren[1] is not Border b) throw new Exception("Такого не бывает");
            border = b;
        }


        public override Point[][] PinPoints { get {
            double X = base_size + width - EllipseStrokeSize / 2;
            double Y = height / 2;
            double PinWidth = base_size - EllipseSize + PinStrokeSize;
            return new Point[][] {
                new Point[] { new(X, Y), new(X + PinWidth, Y) },
            };
        } }


        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 ? "Lime" : "#F08080"));
        }

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


        public override object Export() {
            return new Dictionary<string, object> {
                ["id"] = TypeId,
                ["pos"] = GetPos(),
                ["size"] = GetBodySize(),
                ["state"] = my_state
            };
        }

        public override void Import(Dictionary<string, object> dict) {
            if (!@dict.TryGetValue("pos", out var @value)) { Log.Write("pos-запись элемента не обнаружен"); return; }
            if (@value is not Point @pos) { Log.Write("Неверный тип pos-записи элемента: " + @value); return; }
            Move(@pos);

            if (!@dict.TryGetValue("size", out var @value2)) { Log.Write("size-запись элемента не обнаружен"); return; }
            if (@value2 is not Size @size) { Log.Write("Неверный тип size-записи элемента: " + @value2); return; }
            Resize(@size, false);

            if (!@dict.TryGetValue("state", out var @value3)) { Log.Write("state-запись элемента не обнаружен"); return; }
            if (@value3 is not bool @state) { Log.Write("Неверный тип state-записи элемента: " + @value3); return; }
            my_state = @state;
            if (my_state) border.Background = new SolidColorBrush(Color.Parse("#F08080"));
        }
    }
}
```

## Views/Shapes/XOR_2.axaml.cs

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

namespace LogicSimulator.Views.Shapes {
    public partial class XOR_2: GateBase, IGate, INotifyPropertyChanged {
        public override int TypeId => 3;

        public override int InputCount => 2;
        public override int OutputCount => 1;
        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;

        protected override void Init() {
            height = 30 * 3;
            InitializeComponent();
            DataContext = this;
        }


        public override Point[][] PinPoints { get {
            double X = EllipseSize - EllipseStrokeSize / 2;
            double X2 = base_size + width - EllipseStrokeSize / 2;
            double R = BodyRadius.TopLeft;
            double Y_s = R, Y_m = height / 2, Y_e = height - Y_s;
            double min = EllipseSize + BaseFraction * 2;
            double Y = Y_s + (Y_e - Y_s) / 4;
            double Y2 = Y_s + (Y_e - Y_s) / 4 * 3;
            if (Y2 - Y < min) { Y = Y_m - min / 2; Y2 = Y_m + min / 2; }
            double PinWidth = base_size - EllipseSize + PinStrokeSize;
            return new Point[][] {
                new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход
                new Point[] { new(X, Y2), new(X + PinWidth, Y2) }, // Второй вход
                new Point[] { new(X2, Y_m), new(X2 + PinWidth, Y_m) }, // Единственный выход
            };
        } }


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

## Views/Shapes/LightBulb.axaml.cs

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

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

    public override int InputCount => 1;
    public override int OutputCount => 0;
    public override UserControl GetSelf() => this;
    protected override IGate GetSelfI => this;

    protected override void Init() {
        width = 30 * 2.5;
        height = 30 * 2.5;
        InitializeComponent();
        DataContext = this;
    }

    readonly Border border;
    public LightBulb(): base() {
        if (LogicalChildren[0].LogicalChildren[1] is not Border b) throw new Exception("Такого не бывает");
        border = b;
    }


    public override Point[][] PinPoints { get {
        double X = EllipseSize - EllipseStrokeSize / 2;
        double Y = height / 2;
        double PinWidth = base_size - EllipseSize + PinStrokeSize;
        return new Point[][] {
            new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Единственный вход
        };
    } }


    readonly SolidColorBrush ColorA = new(Color.Parse("#00ff00")); // On
    readonly SolidColorBrush ColorB = new(Color.Parse("#F08080")); // Off
    public void InnerLogic(ref bool[] ins, ref bool[] outs) {
        var value = ins[0];
        Dispatcher.UIThread.InvokeAsync(() => {
            border.Background = value ? ColorA : ColorB;
        });

    }
  }
}
```

## Views/Shapes/Demul.axaml.cs

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

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

        public override int InputCount => 3;
        public override int OutputCount => 4;
        public override UserControl GetSelf() => this;
        protected override IGate GetSelfI => this;

        protected override void Init()
        {
            height = 30 * 4;
            InitializeComponent();
            DataContext = this;
        }


        public override Point[][] PinPoints
        {
            get
            {
                double X = EllipseSize - EllipseStrokeSize / 2;
                double X2 = base_size + width - EllipseStrokeSize / 2;
                double R = BodyRadius.TopLeft;
                double Y_s = R, Y_m = height / 2, Y_e = height - Y_s;
                double min = EllipseSize + BaseFraction * 2;

                double Y = Y_s + (Y_e - Y_s) / 8;
                double Y2 = Y_s + (Y_e - Y_s) / 8 * 3;
                double Y3 = Y_s + (Y_e - Y_s) / 8 * 5;
                double Y4 = Y_s + (Y_e - Y_s) / 8 * 7;
                if (Y2 - Y < min) { Y = Y_m - min / 2 * 3; Y2 = Y_m - min / 2; Y3 = Y_m + min / 2; Y4 = Y_m + min / 2 * 3; }
                double PinWidth = base_size - EllipseSize + PinStrokeSize;
                return new Point[][] {
                new Point[] { new(X, Y), new(X + PinWidth, Y) }, // Первый вход
                new Point[] { new(X, Y3), new(X + PinWidth, Y3) }, // Второй вход
                new Point[] { new(X, Y4), new(X + PinWidth, Y4) }, // Третий вход
                new Point[] { new(X2, Y), new(X2 + PinWidth, Y) }, // Первый выход
                new Point[] { new(X2, Y2), new(X2 + PinWidth, Y2) }, // Второй выход
                new Point[] { new(X2, Y3), new(X2 + PinWidth, Y3) }, // Третий выход
                new Point[] { new(X2, Y4), new(X2 + PinWidth, Y4) }, // Четвёртый выход
            };
            }
        }



        public void InnerLogic(ref bool[] ins, ref bool[] outs)
        {
            bool a = ins[0], b = ins[1], c = ins[2];
            int num = (b ? 1 : 0) + (c ? 2 : 0);
            outs[0] = num == 0 && a;
            outs[1] = num == 1 && a;
            outs[2] = num == 2 && a;
            outs[3] = num == 3 && a;
        }
    }
}
```

## Views/Shapes/AND_2.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="{Binding UC_Width}" d:DesignHeight="{Binding UC_Height}"
        Width="{Binding UC_Width}" Height="{Binding UC_Height}"
                             x:Class="LogicSimulator.Views.Shapes.AND_2"
                             Tag="Gate">

         <Canvas Tag="Gate">
                <Line Tag="Pin" StartPoint="{Binding PinPoints[0][0]}" EndPoint="{Binding PinPoints[0][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                <Line Tag="Pin" StartPoint="{Binding PinPoints[1][0]}" EndPoint="{Binding PinPoints[1][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                <Line Tag="Pin" StartPoint="{Binding PinPoints[2][0]}" EndPoint="{Binding PinPoints[2][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                <Border Tag="Body" Margin="{Binding BodyMargin}" Background="#DC143C"
BorderThickness="{Binding BodyStrokeSize}" BorderBrush="#8B0000" Width="{Binding BodyWidth}" Height="{Binding
BodyHeight}" CornerRadius="{Binding BodyRadius}">
                        <Panel>
                                <TextBlock Tag="Body" FontSize="{Binding FontSizze}"
HorizontalAlignment="Center" FontWeight="Bold" VerticalAlignment="Center" Foreground="White">AND</TextBlock>
                                <Image Tag="Resizer" Width="32" VerticalAlignment="Bottom"
HorizontalAlignment="Right" Margin="{Binding ImageMargins[0]}" Height="32"
Source="avares://LogicSimulator/Assets/Resizer.png"></Image>
                        </Panel>
                </Border>
                <Ellipse Tag="In" Margin="{Binding EllipseMargins[0]}" Width="{Binding EllipseSize}"
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
                <Ellipse Tag="In" Margin="{Binding EllipseMargins[1]}" Width="{Binding EllipseSize}"
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
                <Ellipse Tag="Out" Margin="{Binding EllipseMargins[2]}" Width="{Binding EllipseSize}"
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
        </Canvas>
</UserControl>
```

## Views/Shapes/LightBulb.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="{Binding UC_Width}" d:DesignHeight="{Binding UC_Height}"
        Width="{Binding UC_Width}" Height="{Binding UC_Height}"
                             x:Class="LogicSimulator.Views.Shapes.LightBulb"
                             Tag="Gate">

         <Canvas Tag="Gate">
                <Line Tag="Pin" StartPoint="{Binding PinPoints[0][0]}" EndPoint="{Binding PinPoints[0][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                <Border Tag="Body" Margin="{Binding BodyMargin}" Background="#F08080"
BorderThickness="{Binding BodyStrokeSize}" BorderBrush="#CD5C5C" Width="{Binding BodyWidth}" Height="{Binding
BodyHeight}" CornerRadius="{Binding BodyRadius}">
                        <Panel>
                                <Image Tag="Resizer" Width="24" VerticalAlignment="Bottom"
HorizontalAlignment="Right" Margin="{Binding ImageMargins[0]}" Height="32"
Source="avares://LogicSimulator/Assets/Resizer.png"></Image>
                        </Panel>
                </Border>
                <Ellipse Tag="In" Margin="{Binding EllipseMargins[0]}" Width="{Binding EllipseSize}"
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
        </Canvas>
</UserControl>
```

## Views/Shapes/NOT.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="{Binding UC_Width}" d:DesignHeight="{Binding UC_Height}"
        Width="{Binding UC_Width}" Height="{Binding UC_Height}"
                             x:Class="LogicSimulator.Views.Shapes.NOT"
                             Tag="Gate">

         <Canvas Tag="Gate">
                  <Line Tag="Pin" StartPoint="{Binding PinPoints[0][0]}" EndPoint="{Binding PinPoints[0][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                  <Line Tag="Pin" StartPoint="{Binding PinPoints[1][0]}" EndPoint="{Binding PinPoints[1][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                  <Border Tag="Body" Margin="{Binding BodyMargin}" Background="#DC143C"
BorderThickness="{Binding BodyStrokeSize}" BorderBrush="#8B0000" Width="{Binding BodyWidth}"
Height="{Binding BodyHeight}" CornerRadius="{Binding BodyRadius}">
                           <Panel>
                                    <TextBlock Tag="Body" FontSize="{Binding FontSizze}"
HorizontalAlignment="Center" VerticalAlignment="Center" FontWeight="Bold"
Foreground="White">NOT</TextBlock>
                                    <Image Tag="Resizer" Width="32" VerticalAlignment="Bottom"
HorizontalAlignment="Right" Margin="{Binding ImageMargins[0]}" Height="32"
Source="avares://LogicSimulator/Assets/Resizer.png"></Image>
                           </Panel>
                  </Border>
                  <Ellipse Tag="In" Margin="{Binding EllipseMargins[0]}" Width="{Binding EllipseSize}"
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
                  <Ellipse Tag="Out" Margin="{Binding EllipseMargins[1]}" Width="{Binding EllipseSize}"
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
         </Canvas>
</UserControl>
```

## Views/Shapes/OR_2.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="{Binding UC_Width}" d:DesignHeight="{Binding UC_Height}"
        Width="{Binding UC_Width}" Height="{Binding UC_Height}"
                             x:Class="LogicSimulator.Views.Shapes.OR_2"
                             Tag="Gate">

         <Canvas Tag="Gate">
                  <Line Tag="Pin" StartPoint="{Binding PinPoints[0][0]}" EndPoint="{Binding PinPoints[0][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                  <Line Tag="Pin" StartPoint="{Binding PinPoints[1][0]}" EndPoint="{Binding PinPoints[1][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                  <Line Tag="Pin" StartPoint="{Binding PinPoints[2][0]}" EndPoint="{Binding PinPoints[2][1]}"
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
                  <Border Tag="Body" Margin="{Binding BodyMargin}" Background="#DC143C"
BorderThickness="{Binding BodyStrokeSize}" BorderBrush="#8B0000" Width="{Binding BodyWidth}"
Height="{Binding BodyHeight}" CornerRadius="{Binding BodyRadius}">
                           <Panel>
                                    <TextBlock Tag="Body" FontSize="{Binding FontSizze}"
HorizontalAlignment="Center" VerticalAlignment="Center" FontWeight="Bold"
Foreground="White">OR</TextBlock>
                                    <Image Tag="Resizer" Width="32" VerticalAlignment="Bottom"
HorizontalAlignment="Right" Margin="{Binding ImageMargins[0]}" Height="32"
Source="avares://LogicSimulator/Assets/Resizer.png"></Image>
                           </Panel>
                  </Border>
                  <Ellipse Tag="In" Margin="{Binding EllipseMargins[0]}" Width="{Binding EllipseSize}"
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
                  <Ellipse Tag="In" Margin="{Binding EllipseMargins[1]}" Width="{Binding EllipseSize}"
```

```
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
			<Ellipse Tag="Out" Margin="{Binding EllipseMargins[2]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
		</Canvas>
</UserControl>
```

## Views/Shapes/Demul.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="{Binding UC_Width}" d:DesignHeight="{Binding UC_Height}"
        Width="{Binding UC_Width}" Height="{Binding UC_Height}"
				x:Class="LogicSimulator.Views.Shapes.Demul"
				Tag="Gate">

		<Canvas Tag="Gate">
			<Line Tag="Pin" StartPoint="{Binding PinPoints[0][0]}" EndPoint="{Binding PinPoints[0][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
			<Line Tag="Pin" StartPoint="{Binding PinPoints[1][0]}" EndPoint="{Binding PinPoints[1][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
			<Line Tag="Pin" StartPoint="{Binding PinPoints[2][0]}" EndPoint="{Binding PinPoints[2][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
			<Line Tag="Pin" StartPoint="{Binding PinPoints[3][0]}" EndPoint="{Binding PinPoints[3][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
			<Line Tag="Pin" StartPoint="{Binding PinPoints[4][0]}" EndPoint="{Binding PinPoints[4][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
			<Line Tag="Pin" StartPoint="{Binding PinPoints[5][0]}" EndPoint="{Binding PinPoints[5][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
			<Line Tag="Pin" StartPoint="{Binding PinPoints[6][0]}" EndPoint="{Binding PinPoints[6][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
			<Border Tag="Body" Margin="{Binding BodyMargin}"  Background="#DC143C" 
BorderThickness="{Binding BodyStrokeSize}" BorderBrush="#8B0000" Width="{Binding BodyWidth}" Height="{Binding 
BodyHeight}" CornerRadius="{Binding BodyRadius}">
				<Panel>
					<TextBlock Tag="Body" FontSize="{Binding FontSizze}" 
HorizontalAlignment="Center" FontWeight="Bold" VerticalAlignment="Center" Foreground="White">Demul</TextBlock>
					<Image Tag="Resizer" Width="32" VerticalAlignment="Bottom" 
HorizontalAlignment="Right" Margin="{Binding ImageMargins[0]}" Height="32" 
Source="avares://LogicSimulator/Assets/Resizer.png"></Image>
				</Panel>
			</Border>
			<Ellipse Tag="In" Margin="{Binding EllipseMargins[0]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
			<Ellipse Tag="In" Margin="{Binding EllipseMargins[1]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
			<Ellipse Tag="In" Margin="{Binding EllipseMargins[2]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
			<Ellipse Tag="Out" Margin="{Binding EllipseMargins[3]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
			<Ellipse Tag="Out" Margin="{Binding EllipseMargins[4]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
			<Ellipse Tag="Out" Margin="{Binding EllipseMargins[5]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
			<Ellipse Tag="Out" Margin="{Binding EllipseMargins[6]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
		</Canvas>
</UserControl>
```

## Views/Shapes/Switch.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="{Binding UC_Width}" d:DesignHeight="{Binding UC_Height}"
        Width="{Binding UC_Width}" Height="{Binding UC_Height}"
				x:Class="LogicSimulator.Views.Shapes.Switch"
				Tag="Gate">
```

```
	<Canvas Tag="Gate">
		<Line Tag="Pin" StartPoint="{Binding PinPoints[0][0]}" EndPoint="{Binding PinPoints[0][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
		<Border Tag="Body" Margin="{Binding BodyMargin}" Background="#F08080" 
BorderThickness="{Binding BodyStrokeSize}" BorderBrush="#CD5C5C" Width="{Binding BodyWidth}" Height="{Binding 
BodyHeight}" CornerRadius="{Binding BodyRadius}" PointerPressed="Press" PointerReleased="Release">
			<Panel>
				<Image Tag="Resizer" Width="32" VerticalAlignment="Bottom" 
HorizontalAlignment="Right" Margin="{Binding ImageMargins[0]}" Height="24" 
Source="avares://LogicSimulator/Assets/Resizer.png"></Image>
			</Panel>
		</Border>
		<Ellipse Tag="Out" Margin="{Binding EllipseMargins[0]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
	</Canvas>
</UserControl>
```

## Views/Shapes/XOR_2.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="{Binding UC_Width}" d:DesignHeight="{Binding UC_Height}"
        Width="{Binding UC_Width}" Height="{Binding UC_Height}"
                     x:Class="LogicSimulator.Views.Shapes.XOR_2"
                     Tag="Gate">

	<Canvas Tag="Gate">
		<Line Tag="Pin" StartPoint="{Binding PinPoints[0][0]}" EndPoint="{Binding PinPoints[0][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
		<Line Tag="Pin" StartPoint="{Binding PinPoints[1][0]}" EndPoint="{Binding PinPoints[1][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
		<Line Tag="Pin" StartPoint="{Binding PinPoints[2][0]}" EndPoint="{Binding PinPoints[2][1]}" 
Stroke="Gray" StrokeThickness="{Binding PinStrokeSize}"/>
		<Border Tag="Body" Margin="{Binding BodyMargin}" Background="#DC143C" 
BorderThickness="{Binding BodyStrokeSize}" BorderBrush="#8B0000" Width="{Binding BodyWidth}" Height="{Binding 
BodyHeight}" CornerRadius="{Binding BodyRadius}">
			<Panel>
				<TextBlock Tag="Body" FontSize="{Binding FontSizze}" 
HorizontalAlignment="Center" VerticalAlignment="Center" FontWeight="Bold" Foreground="White">XOR</TextBlock>
				<Image Tag="Resizer" Width="32" VerticalAlignment="Bottom" 
HorizontalAlignment="Right" Margin="{Binding ImageMargins[0]}" Height="32" 
Source="avares://LogicSimulator/Assets/Resizer.png"></Image>
			</Panel>
		</Border>
		<Ellipse Tag="In" Margin="{Binding EllipseMargins[0]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
		<Ellipse Tag="In" Margin="{Binding EllipseMargins[1]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
		<Ellipse Tag="Out" Margin="{Binding EllipseMargins[2]}" Width="{Binding EllipseSize}" 
Height="{Binding EllipseSize}" Stroke="Gray" StrokeThickness="{Binding EllipseStrokeSize}" Fill="#808080"/>
	</Canvas>
</UserControl>
```

## Views/LauncherWindow.axaml

```
<Window xmlns="https://github.com/avaloniaui"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:vm="using:LogicSimulator.ViewModels"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      mc:Ignorable="d" d:DesignWidth="500" d:DesignHeight="800"
		Width="500" Height="800"
      x:Class="LogicSimulator.Views.LauncherWindow"
      Icon="/Assets/redstone_logo.ico"
      Title="LogicSimulator"
		Padding="8" Background="#F08080">

  <Design.DataContext>
```

```
        <vm:LauncherWindowViewModel/>
    </Design.DataContext>

	<Window.Styles>
		<Style Selector="ListBoxItem">
			<Setter Property="Padding" Value="0"/>
			<Setter Property="Margin" Value="0 0 0 10"/>
		</Style>
		<Style Selector="Button">
			<Setter Property="BorderThickness" Value="2"/>
			<Setter Property="Background" Value="#B22222"/>
			<Setter Property="Foreground" Value="White"/>
			<Setter Property="CornerRadius" Value="2"/>
			<Setter Property="Padding" Value="10"/>
			<Setter Property="FontSize" Value="32"/>
			<Setter Property="HorizontalAlignment" Value="Center"/>
		</Style>
		<Style Selector="Border.b">
			<Setter Property="BorderThickness" Value="4"/>
			<Setter Property="BorderBrush" Value="#DC143C"/>
			<Setter Property="CornerRadius" Value="2"/>
			<Setter Property="Padding" Value="10"/>
		</Style>
		<Style Selector="TextBlock.tb">
			<Setter Property="Margin" Value="5"/>
			<Setter Property="Padding" Value="4"/>
			<Setter Property="FontSize" Value="32"/>
			<Setter Property="HorizontalAlignment" Value="Center"/>
		</Style>
		<Style Selector="ListBox.lb">
			<Setter Property="Background" Value="White"/>
		</Style>
	</Window.Styles>

	<Grid RowDefinitions="auto,auto,*,auto">
		<Button Command="{Binding Create}" FontWeight="Bold">Создать новый проект</Button>
		<TextBlock Grid.Row="1" Classes="tb">Перечень проектов:</TextBlock>
		<ListBox Grid.Row="2" Classes="lb" Items="{Binding ProjectList}" DoubleTapped="DTapped"
Padding="10">
			<ListBox.ItemTemplate>
				<DataTemplate>
					<Border Classes="b">
						<TextBlock Text="{Binding}" Tag="{Binding}"/>
					</Border>
				</DataTemplate>
			</ListBox.ItemTemplate>
		</ListBox>
		<Button Grid.Row="3" FontWeight="Bold" Command="{Binding Exit}">Выход</Button>
	</Grid>
</Window>
```

## Views/MainWindow.axaml

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="using:LogicSimulator.ViewModels"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="1400" d:DesignHeight="800"
		Width="1400" Height="800"
        x:Class="LogicSimulator.Views.MainWindow"
        Icon="/Assets/redstone_logo.ico"
        Title="LogicSimulator">

    <Design.DataContext>
        <vm:MainWindowViewModel/>
    </Design.DataContext>

	<Window.Styles>
		<Style Selector="ListBoxItem">
```

```
				<Setter Property="Padding" Value="0"/>
			</Style>
			<Style Selector="Border.b">
				<Setter Property="BorderThickness" Value="3"/>
				<Setter Property="BorderBrush" Value="#8B0000"/>
			</Style>
			<Style Selector="TextBox">
				<Setter Property="Margin" Value="-5"/>
				<Setter Property="Padding" Value="4"/>
				<Setter Property="MinHeight" Value="0"/>
			</Style>
			<Style Selector="ListBox.cl">
				<Setter Property="Background" Value="#fce8e8"/>
			</Style>

	</Window.Styles>

	<DockPanel>
		<Menu DockPanel.Dock="Top">
			<MenuItem Header="Файл">
				<MenuItem Header="Создать" Command="{Binding Comm}"
CommandParameter="Create"/>
				<MenuItem Header="Открыть проект" Command="{Binding Comm}"
CommandParameter="Open"/>
				<MenuItem Header="Сохранить текущую схему" Command="{Binding Comm}"
CommandParameter="Save"/>
				<MenuItem Header="Выйти" Command="{Binding Comm}"
CommandParameter="Exit"/>
			</MenuItem>
		</Menu>

		<Grid ColumnDefinitions="*,5*">
			<Border Classes="b">
			<ListBox Classes="cl" Items="{Binding ItemTypes}" SelectedIndex="{Binding SelectedItem}">
				<ListBox.ItemTemplate>
					<DataTemplate>
						<ContentControl Content="{Binding}"/>
					</DataTemplate>
				</ListBox.ItemTemplate>
			</ListBox>
			</Border>
			<Panel Grid.Column="1">
				<TextBlock Text="{Binding Logg}" Background="White"/>
				<Canvas Tag="Scene" Name="Canvas" Background="#0000"/>
			</Panel>

		</Grid>

	</DockPanel>
</Window>
```