Computing Write Up

# Analysis

## The problem

The person I designed the program for is myself as an Electronics student. As such, there was no particular communication between programmer and client. I wished to practice my programming, and a topic that caught my interest was logic circuit simulations – I had the start of an idea of how I would design it, and so I began to program it. I am advanced enough in my programming and was sufficiently progressed through the project, that when it came time to focus on the coursework, I decided to keep going with the circuit sim.

## The current system

As an Electronics student, we use ISIS for our circuit simulations, to design circuits before, or in lieu of, building them in real life. ISIS has a large library of existing components, as well as supporting analogue signals. However, it does not support custom components.

### Problems

1. ISIS does not support custom components. There is no way to create a component that can be placed and reused like any other unless it is already within the library, possibly excluding writing code to control it.
2. ISIS has very little customisability of components. Only components such as resistors and capacitors have any form of customisability.
3. I have no understanding of ISIS under the hood. I merely know how to use it as any consumer would. This is not a true problem with the program, but a source of my original motivation to create an alternative.

## Objectives

1. A way to design and simulate a circuit.
2. A way to create a circuit, then later reuse it as a single component.
3. A way to customise components.

# Design

## Namespaces

* CircuitMaker is the namespace for the whole program.
  + Basics contains the basic classes such as Pin, Pos, Board, and others, and also contains the interfaces implemented by the Components.
  + Components contains all my predefined components.
  + GUI contains all the GUI code, including GUIForm, ComponentSelectionForm, and RenameBoardForm
    - Settings contains the whole component settings system, including the GUI, found in SettingsForm.
    - ExtApp contains the whole GUI for editing the external appearance of a board, found in ExtAppEditorForm.

## Forms

### GUIForm

GUIForm is the primary form, containing a selection of options in the drop down menu found at the top of the window that many apps have, and the editor itself (the class known as Builder).

### ComponentSelectionForm

ComponentSelectionForm is a simple form that contains a list of all hard-coded components (found in the CircuitMaker.Components namespace). It allows for selecting one of them and placing them in the editor.

### ExtAppEditorForm

ExtAppEditorForm is a form for designing the external appearance of the component that will be created when selecting a file from the Create Board Component option. It allows for resizing of the component, positioning of all the pins, and positioning and scaling of the graphics of all the IGraphicalComponents within.

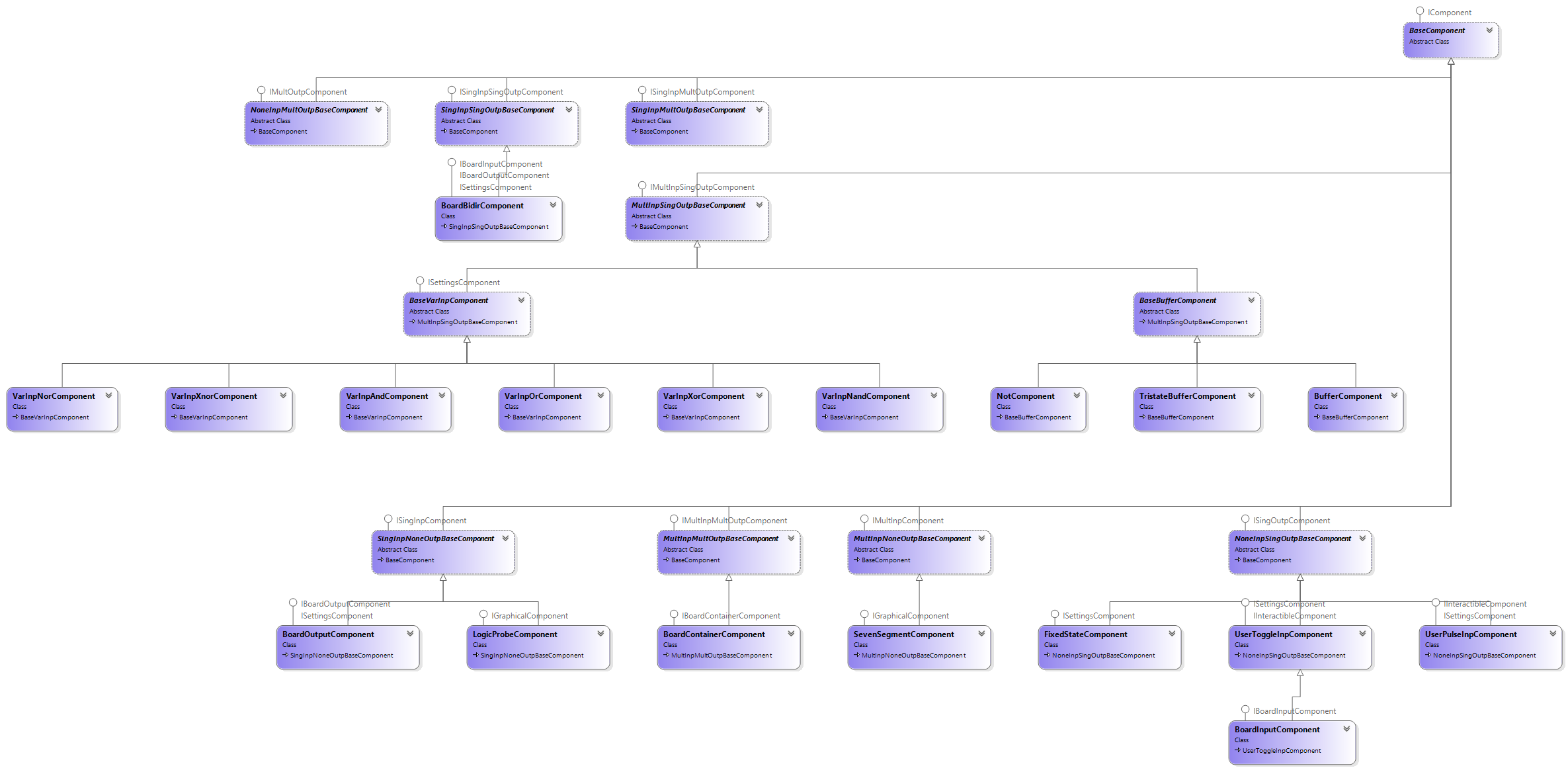
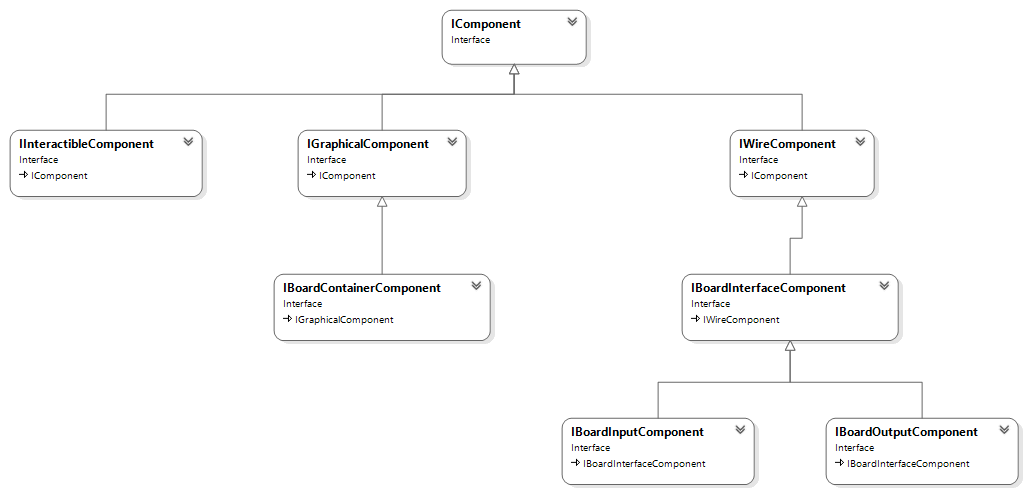
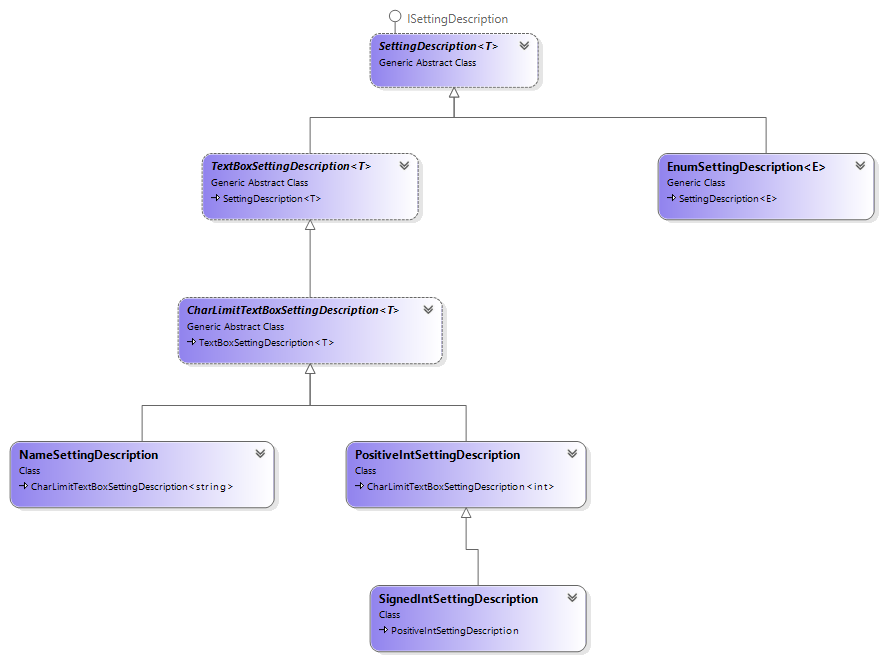
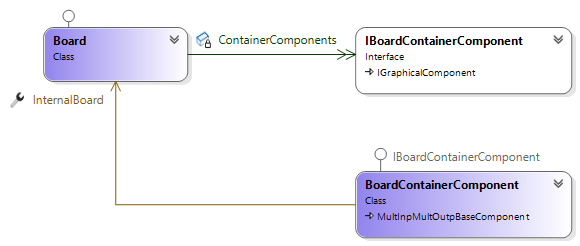
### RenameBoardForm

RenameBoardForm is a simple form with a text box to enter the new name of the board.

### SettingsDialog

SettingsDialog is a dynamic form created either when double clicking on a component, or choosing the Open Settings option when left clicking on a component, as long as the component has settings to edit. It creates a label and asks the component for an input control for each setting, and then arranges them vertically. The component retains a reference to all of these input controls, and then alters its own fields according to the value in the controls if the OK button is used to close the SettingsDialog.

## Classes



## File Save

Files are written straight to binary. The encoding is as follows:

### File

* Top level Board.
* Sub-Boards (4-bit length followed by boards end to end).

### Board

* Name (string: 1 bit length followed by characters).
* External Size (two 4-bit numbers, width then height).
* Components (4-bit length followed by components end to end).
* Wires (4-bit length followed by wires end to end).

### Component

* ID (string).
* Details (string).
* Position.
* Rotation (member of Rotation enumeration converted into string format and stored as such)

### Wire

* Position of one end.
* Position of the other end.

### Position

* X value (4-bit integer).
* Y value (4-bit integer).

# Testing

## Procedure

1. Construct a simple circuit, with inputs and outputs.
   * Tests placement of components.
   * Tests ComponentSelectionForm.
   * Tests SettingsDialog
   * Tests wire placement.
   * Tests navigation.
2. Simulate the circuit.
   * Tests simulation.
3. Save the circuit.
   * Tests saving files.
4. Load the circuit, edit its external appearance, and save it again.
   * Tests loading files.
   * Tests ExtAppEditor.
   * Tests saving external appearance.
5. Load the circuit into a BoardContainerComponent.
   * Tests BoardContainerComponent.
6. Construct a new circuit using the component containing the previous circuit, and simulate it.
   * Tests simulation of contained boards.
7. Save this new circuit, load it into a BoardContainerComponent itself, use it in a new circuit twice, and simulate it.
   * Tests simulation of recursively contained boards, and copied BoardContainerComponents.

## Results

The results can be found in the TestVid file uploaded alongside this write up.

# Evaluation

1. Boards can be designed and simulated accurately.
2. Custom components can be created as a board, saved, and reused in any board by selecting the file.
3. Programmed components can be customised through the use of the SettingsForm.

There are features of C# I was not aware of when beginning the project that I would have used, but in order to identify them I would have to start over, as it is likely some design decisions would be different. Additionally, due to the complexity of the program, there are many bugs that I never got the chance to fix – hopefully there are no breaking issues hiding. I haven’t had the chance to get someone to try and use it.

Bugs I have noticed are:

* The system for keeping track of which pins a component is attached to is non-functional.
  + I could fix this by removing the complex, count-them-in count-them-out system I added to allow for before a BoardContainerComponent knows what it contains, and editing the function used to get all the pins to return an empty list instead of crashing if the BoardContainerComponent doesn’t know what it contains.
* Bidirectional connections are broken.
  + The DefaultDetails static field contains the wrong value, which means when adding a BIDIR component throught the ComponentSelectionForm, it can’t properly initialise it. This is easily fixed by editing it the right value.

Thinking about it as an Electronics student instead of the programmer, I see it as fully functional for my purposes. In fact, I have successfully recreated my Electronics coursework in the program, and it behaved exactly as I expected it to.

# Code

My full code can be found at <https://github.com/LastedApple3/CircuitMaker/tree/exam-submission>

## Basics

### TransformRestorer

class TransformRestorer : IDisposable

{

private Graphics SavedGraphics;

private Matrix SavedMatrix;

public TransformRestorer(Graphics graphics)

{

SavedGraphics = graphics;

SavedMatrix = graphics.Transform;

}

public void Dispose()

{

SavedGraphics.Transform = SavedMatrix;

}

}

### RectangleFExtensions

static class RectangleFExtensions

{

public static int dp = 5;

private static float Round(float val)

{

return (float)Math.Round(val, dp);

}

public static RectangleF Round(this RectangleF rect)

{

return new RectangleF(Round(rect.X), Round(rect.Y), Round(rect.Width), Round(rect.Height));

}

}

### ByteEncoding

public class ByteEncoding : Encoding

{

public override int GetByteCount(char[] chars, int index, int count)

{

return count;

}

public override int GetBytes(char[] chars, int charIndex, int charCount, byte[] bytes, int byteIndex)

{

for (int i = 0; i < charCount; i++)

{

bytes[byteIndex + i] = (byte)chars[charIndex + i];

}

return charCount;

}

public override int GetCharCount(byte[] bytes, int index, int count)

{

return count;

}

public override int GetChars(byte[] bytes, int byteIndex, int byteCount, char[] chars, int charIndex)

{

for (int i = 0; i < byteCount; i++)

{

chars[charIndex + i] = (char)bytes[byteIndex + i];

}

return byteCount;

}

public override int GetMaxByteCount(int charCount)

{

return charCount;

}

public override int GetMaxCharCount(int byteCount)

{

return byteCount;

}

private static Encoding byteEncoding = null;

public static Encoding Byte

{

get

{

if (byteEncoding == null)

{

byteEncoding = new ByteEncoding();

}

return byteEncoding;

}

}

}

### PlacementException

public class PlacementException : Exception

{

public PlacementException(string desc) : base(desc) { }

}

### ReadWriteImplementation

public static class ReadWriteImplementation

{

public static void Write<T>(this BinaryWriter bw, T enumVal) where T : Enum

{

bw.Write(Enum.GetName(typeof(T), enumVal));

}

public static T ReadEnum<T>(this BinaryReader br) where T : Enum

{

return (T)Enum.Parse(typeof(T), br.ReadString());

}

public static void Write(this BinaryWriter bw, Pos pos)

{

bw.Write(pos.X);

bw.Write(pos.Y);

}

public static Pos ReadPos(this BinaryReader br)

{

return new Pos(br.ReadInt32(), br.ReadInt32());

}

public static void Write(this BinaryWriter bw, Wire wire)

{

bw.Write(wire.Pos1);

bw.Write(wire.Pos2);

}

public static Wire ReadWire(this BinaryReader br, Board board)

{

return new Wire(br.ReadPos(), br.ReadPos(), board);

}

public static void Write(this BinaryWriter bw, IComponent comp)

{

bw.Write(comp.GetComponentID());

bw.Write(comp.GetComponentDetails());

bw.Write(comp.GetComponentPos());

bw.Write(comp.GetComponentRotation());

if (comp is IGraphicalComponent graphicalComp)

{

bw.Write(graphicalComp.GetGraphicalElementScale());

Point? graphicalLoc = graphicalComp.GetGraphicalElementLocation();

bw.Write(graphicalLoc.HasValue);

if (graphicalLoc.HasValue)

{

bw.Write(graphicalLoc.Value.X);

bw.Write(graphicalLoc.Value.Y);

}

}

if (comp is IBoardInterfaceComponent interfaceComp)

{

Board.InterfaceLocation intLoc = interfaceComp.GetInterfaceLocation();

bw.Write(intLoc.Side);

bw.Write(intLoc.Distance);

}

}

public static IComponent ReadComponent(this BinaryReader br, Board board)

{

string compType = br.ReadString();

if (Constructors.TryGetValue(compType, out Func<string, IComponent> compFunc))

{

IComponent comp = compFunc(br.ReadString());

Pos pos = br.ReadPos();

Rotation rot = br.ReadEnum<Rotation>();

if (comp is IGraphicalComponent graphicalComp)

{

graphicalComp.SetGraphicalElementScale(br.ReadSingle());

if (br.ReadBoolean())

{

graphicalComp.SetGraphicalElementLocation(new Point(br.ReadInt32(), br.ReadInt32()));

}

}

if (comp is IBoardInterfaceComponent interfaceComp)

{

interfaceComp.SetInterfaceLocation(new Board.InterfaceLocation(br.ReadEnum<Board.InterfaceLocation.SideEnum>(), br.ReadInt32()));

}

comp.Place(pos, rot, board);

return comp;

}

throw new PlacementException($"Could not find builtin component of type {compType}");

}

public static void Write(this BinaryWriter bw, Board board)

{

board.SimplifyWires();

WriteBoardBasic(bw, board);

Board[] boards = board.GetBoardList().Where(thisBoard => thisBoard.Name != board.Name).ToArray();

bw.Write(boards.Length);

foreach (Board thisBoard in boards)

{

WriteBoardBasic(bw, thisBoard);

}

}

private static void WriteBoardBasic(BinaryWriter bw, Board board)

{

bw.Write(board.Name);

bw.Write(board.ExternalSize.Width);

bw.Write(board.ExternalSize.Height);

IComponent[] comps = board.GetComponents();

Wire[] wires = board.GetAllWires();

bw.Write(comps.Length);

foreach (IComponent comp in comps)

{

bw.Write(comp);

}

bw.Write(wires.Length);

foreach (Wire wire in wires)

{

bw.Write(wire);

}

}

public static Board ReadBoard(this BinaryReader br)

{

Board topLevelBoard = ReadBoardBasic(br);

int boardCount = br.ReadInt32();

Board[] boards = new Board[boardCount];

for (int i = 0; i < boardCount; i++)

{

boards[i] = ReadBoardBasic(br);

}

topLevelBoard.SupplyInternalBoards(boards);

return topLevelBoard;

}

private static Board ReadBoardBasic(BinaryReader br)

{

Board board = new Board(br.ReadString(), new Size(br.ReadInt32(), br.ReadInt32()));

int compCount = br.ReadInt32();

for (int i = 0; i < compCount; i++)

{

br.ReadComponent(board);

}

int wireCount = br.ReadInt32();

for (int i = 0; i < wireCount; i++)

{

br.ReadWire(board);

}

return board;

}

public static Dictionary<string, Func<string, IComponent>> Constructors = new Dictionary<string, Func<string, IComponent>>();

public static Dictionary<string, string> DefaultDetails = new Dictionary<string, string>();

}

### DefaultDictionary

class DefaultDictionary<TKey, TValue> : Dictionary<TKey, TValue>

{

protected readonly Func<TValue> BlindGenerator;

protected readonly Func<TKey, TValue> KeyBasedGenerator;

protected readonly Func<DefaultDictionary<TKey, TValue>, TValue> DictBasedGenerator;

protected readonly Func<DefaultDictionary<TKey, TValue>, TKey, TValue> DictAndKeyBasedGenerator;

protected readonly Func<TKey, bool> KeyKeepChecker;

protected readonly Func<TValue, bool> ValKeepChecker;

public DefaultDictionary()

{

ValKeepChecker = val => val.Equals(default(TValue));

}

public DefaultDictionary(Func<TValue> generator)

{

BlindGenerator = generator;

}

public DefaultDictionary(Func<TKey, TValue> generator)

{

KeyBasedGenerator = generator;

}

public DefaultDictionary(Func<DefaultDictionary<TKey, TValue>, TValue> generator)

{

DictBasedGenerator = generator;

}

public DefaultDictionary(Func<DefaultDictionary<TKey, TValue>, TKey, TValue> generator)

{

DictAndKeyBasedGenerator = generator;

}

public DefaultDictionary(Func<TValue> generator, Func<TValue, bool> valKeepChecker)

{

BlindGenerator = generator;

ValKeepChecker = valKeepChecker;

}

public DefaultDictionary(Func<TKey, TValue> generator, Func<TValue, bool> valKeepChecker)

{

KeyBasedGenerator = generator;

ValKeepChecker = valKeepChecker;

}

public DefaultDictionary(Func<DefaultDictionary<TKey, TValue>, TValue> generator, Func<TValue, bool> valKeepChecker)

{

DictBasedGenerator = generator;

ValKeepChecker = valKeepChecker;

}

public DefaultDictionary(Func<DefaultDictionary<TKey, TValue>, TKey, TValue> generator, Func<TValue, bool> valKeepChecker)

{

DictAndKeyBasedGenerator = generator;

ValKeepChecker = valKeepChecker;

}

public DefaultDictionary(Func<TValue> generator, Func<TKey, bool> keyKeepChecker)

{

BlindGenerator = generator;

KeyKeepChecker = keyKeepChecker;

}

public DefaultDictionary(Func<TKey, TValue> generator, Func<TKey, bool> keyKeepChecker)

{

KeyBasedGenerator = generator;

KeyKeepChecker = keyKeepChecker;

}

public DefaultDictionary(Func<DefaultDictionary<TKey, TValue>, TValue> generator, Func<TKey, bool> keyKeepChecker)

{

DictBasedGenerator = generator;

KeyKeepChecker = keyKeepChecker;

}

public DefaultDictionary(Func<DefaultDictionary<TKey, TValue>, TKey, TValue> generator, Func<TKey, bool> keyKeepChecker)

{

DictAndKeyBasedGenerator = generator;

KeyKeepChecker = keyKeepChecker;

}

public void TrimDown()

{

HashSet<TKey> removeKeys = new HashSet<TKey>();

if (ValKeepChecker != null)

{

foreach (TKey key in Keys)

{

if (!ValKeepChecker(base[key]))

{

removeKeys.Add(key);

}

}

} else if (KeyKeepChecker != null)

{

foreach (TKey key in Keys)

{

if (!KeyKeepChecker(key))

{

removeKeys.Add(key);

}

}

}

removeKeys.Select(Remove);

}

public new TValue this[TKey key]

{

get

{

if (!ContainsKey(key))

{

if (BlindGenerator != null)

{

Add(key, BlindGenerator());

} else if (KeyBasedGenerator != null)

{

Add(key, KeyBasedGenerator(key));

} else if (DictBasedGenerator != null)

{

Add(key, DictBasedGenerator(this));

} else if (DictAndKeyBasedGenerator != null)

{

Add(key, DictAndKeyBasedGenerator(this, key));

} else

{

Add(key, default);

}

}

return base[key];

}

set

{

base[key] = value;

TrimDown();

}

}

}

### InstanceTracker

public class InstanceTracker<T> where T : class, InstanceTracker<T>.ITrackable

{

Dictionary<uint, WeakReference<T>> dict = new Dictionary<uint, WeakReference<T>>();

public InstanceTracker()

{

}

private uint GetFirstEmptyID()

{

TrimDown();

uint id = 0;

while (dict.ContainsKey(id))

{

id++;

}

return id;

}

public void TrimDown()

{

bool loop;

do

{

loop = false;

foreach (uint id in dict.Keys)

{

if (!dict[id].TryGetTarget(out T obj))

{

dict.Remove(id);

loop = true;

break;

}

}

} while (loop);

}

private void Add(uint id, T obj)

{

obj.SetTrackingID(id);

dict.Add(id, new WeakReference<T>(obj));

}

public void Add(T obj)

{

Add(GetFirstEmptyID(), obj);

}

public T this[uint id]

{

get

{

if (dict[id].TryGetTarget(out T retObj))

{

return retObj;

}

return null;

}

set

{

TrimDown();

if (dict.ContainsKey(id))

{

dict[id].SetTarget(value);

} else

{

Add(id, value);

}

}

}

}

#### ITrackable

public interface ITrackable

{

void SetTrackingID(uint id);

uint GetTrackingID();

}

### Pos

public readonly struct Pos

{

public readonly int X;

public readonly int Y;

public Pos(int X, int Y)

{

this.X = X;

this.Y = Y;

}

public bool Equals(Pos other)

{

return X == other.X && Y == other.Y;

}

public Pos Add(int X, int Y)

{

return new Pos(this.X + X, this.Y + Y);

}

public Pos Add(Pos pos)

{

return new Pos(X + pos.X, Y + pos.Y);

}

public override string ToString()

{

return $"({X}, {Y})";

}

public static bool operator ==(Pos pos1, Pos pos2)

{

return pos1.X == pos2.X && pos1.Y == pos2.Y;

}

public static bool operator !=(Pos pos1, Pos pos2)

{

return pos1.X != pos2.X || pos1.Y != pos2.Y;

}

public override bool Equals(object obj)

{

if (obj is Pos pos)

{

return this == pos;

}

return base.Equals(obj);

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public Pos Rotate(Rotation rotation)

{

if (rotation == Rotation.CLOCKWISE)

{

return new Pos(-Y, X);

}

else if (rotation == Rotation.HALF)

{

return new Pos(-X, -Y);

}

else if (rotation == Rotation.ANTICLOCKWISE)

{

return new Pos(Y, -X);

}

else

{

return this;

}

}

public Point ToPoint()

{

return new Point(X, Y);

}

public static Pos FromPoint(Point point)

{

return new Pos(point.X, point.Y);

}

}

### Rotation

public enum Rotation

{

ZERO = 0, CLOCKWISE = 90, HALF = 180, ANTICLOCKWISE = 270

}

### RotationExtensions

static class RotationExtensions

{

public static Rotation AddRotation(this Rotation rot1, Rotation rot2)

{

return (Rotation)(((int)rot1 + (int)rot2) % 360);

}

}

### Wire

public class Wire

{

public readonly Pos Pos1;

public readonly Pos Pos2;

private readonly Board board;

public bool IsPlaced { get; private set; }

public Pin Pin1

{

get

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

return board[Pos1];

}

}

public Pin Pin2

{

get

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

return board[Pos2];

}

}

public Wire(Pos pos1, Pos pos2, Board board)

{

Pos1 = pos1;

Pos2 = pos2;

this.board = board;

board.AddWire(this);

IsPlaced = true;

}

public void OnWireUpdate()

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

Pin1.SetState(Pin2.GetStateForWire());

Pin2.SetState(Pin1.GetStateForWire());

}

public bool TrySplitWire(Pos pos)

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

if (Collision(pos))

{

new Wire(Pos1, pos, board);

new Wire(Pos2, pos, board);

Remove();

return true;

}

return false;

}

public void Remove()

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

board.RemoveWire(this);

IsPlaced = false;

}

public override string ToString()

{

return IsPlaced ? $"[{Pos1}, {Pos2}]" : "Removed Wire";

}

public static bool operator ==(Wire wire1, Wire wire2)

{

if (wire1 is null && wire2 is null)

{

return true;

} else if (wire1 is null || wire2 is null)

{

return false;

}

if (wire1.board != wire2.board)

{

return false;

}

return (wire1.Pos1 == wire2.Pos1 && wire1.Pos2 == wire2.Pos2) || (wire1.Pos1 == wire2.Pos2 && wire1.Pos2 == wire2.Pos1);

}

public static bool operator !=(Wire wire1, Wire wire2)

{

if (wire1 is null && wire2 is null)

{

return false;

} else if (wire1 is null || wire2 is null)

{

return true;

}

if (wire1.board != wire2.board)

{

return true;

}

return (wire1.Pos1 != wire2.Pos1 || wire1.Pos2 != wire2.Pos2) && (wire1.Pos1 != wire2.Pos2 || wire1.Pos2 != wire2.Pos1);

}

public override bool Equals(object obj)

{

if (obj is Wire wire)

{

return this == wire;

}

return base.Equals(obj);

}

public override int GetHashCode()

{

return Pos1.GetHashCode() + Pos2.GetHashCode();

}

public Rectangle Bounds()

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

return Rectangle.FromLTRB(

Math.Min(Pos1.X, Pos2.X),

Math.Min(Pos1.Y, Pos2.Y),

Math.Max(Pos1.X, Pos2.X),

Math.Max(Pos1.Y, Pos2.Y));

}

public RectangleF InflatedBounds()

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

RectangleF bounds = Bounds();

bounds.Inflate(0.25F, 0.25F);

return bounds;

}

public bool IsVert()

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

return Pos1.X == Pos2.X;

}

public bool IsHori()

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

return Pos1.Y == Pos2.Y;

}

public bool Collision(Pos pos)

{

if (!IsPlaced)

{

throw new ObjectDisposedException("This wire has already been removed.");

}

Rectangle bounds = Bounds();

if (IsVert())

{

return bounds.Top < pos.Y && pos.Y < bounds.Bottom && pos.X == Pos1.X;

}

if (IsHori())

{

return bounds.Left < pos.X && pos.X < bounds.Right && pos.Y == Pos1.Y;

}

return bounds.Contains(new Point(pos.X, pos.Y));

}

}

### ColourScheme

public struct ColourScheme

{

public Color Background, ComponentBackground, ComponentEdge, Wire, WireFloating, WireLow, WirePulledLow, WireHigh, WirePulledHigh, WireIllegal, Grid, Selection;

public Color GetWireColour(Pin.State state)

{

switch (state)

{

case Pin.State.FLOATING: return WireFloating;

case Pin.State.LOW: return WireLow;

case Pin.State.PULLEDLOW: return WirePulledLow;

case Pin.State.HIGH: return WireHigh;

case Pin.State.PULLEDHIGH: return WirePulledHigh;

case Pin.State.ILLEGAL: return WireIllegal;

default: return Wire;

}

}

}

### IComponent

public interface IComponent

{

void Place(Pos pos, Board board);

void Place(Pos pos, Rotation rotation, Board board);

void Remove();

bool IsPlaced();

void Tick();

void ResetToDefault();

Pos[] GetAllPinOffsets();

Pos[] GetAllPinPositions();

Pos[] GetAllUniquePinPositions();

Pos GetComponentPos();

Rotation GetComponentRotation();

Matrix GetRenderMatrix();

Board GetComponentBoard();

string GetComponentID();

string GetComponentDetails();

RectangleF GetComponentBounds();

RectangleF GetOffsetComponentBounds();

void Render(Graphics graphics, bool simulating, ColourScheme colourScheme);

void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme);

}

### IInteractibleComponent

public interface IInteractibleComponent : IComponent

{

void Interact();

}

### IWireComponent

public interface IWireComponent : IComponent

{

bool TickAgain();

}

### IBoardInterfaceComponent

public interface IBoardInterfaceComponent : IWireComponent

{

string GetComponentName();

void SetComponentName(string compName);

void SetExternalPin(Pin pin);

void RemoveExternalPin();

Board.InterfaceLocation GetInterfaceLocation();

void SetInterfaceLocation(Board.InterfaceLocation location);

}

### IBoardInputComponent

public interface IBoardInputComponent : IBoardInterfaceComponent { }

### IBoardOutputComponent

public interface IBoardOutputComponent : IBoardInterfaceComponent { }

### IGraphicalComponent

public interface IGraphicalComponent : IComponent

{

bool HasGraphics();

void RenderGraphicalElement(Graphics graphics, bool simulating, ColourScheme colourScheme);

RectangleF GetGraphicalElementBounds();

Point? GetGraphicalElementLocation();

void SetGraphicalElementLocation(Point? location);

float GetGraphicalElementScale();

void SetGraphicalElementScale(float scale);

}

### IBoardContainerComponent

public interface IBoardContainerComponent : IGraphicalComponent

{

Rectangle GetShape();

void ResetShape();

string GetInternalBoardName();

Board GetInternalBoard();

void ProvideInternalBoard(Board board);

void PromiseDetails(Action<IBoardContainerComponent> detailProvider);

}

### ComponentExtensions

public static class ComponentExtensions

{

private static RectangleF Scale(RectangleF rect, float scale)

{

rect.X \*= scale;

rect.Y \*= scale;

rect.Width \*= scale;

rect.Height \*= scale;

return rect;

}

private static RectangleF? Offset(RectangleF rect, Point? pos)

{

if (pos.HasValue)

{

rect.Offset(pos.Value);

return rect;

}

return null;

}

public static RectangleF? GetOffsetGraphicalElementBounds<T>(this T comp) where T : IGraphicalComponent

{

return Offset(comp.GetGraphicalElementBounds(), comp.GetGraphicalElementLocation());

}

public static RectangleF GetScaledGraphicalElementBounds<T>(this T comp) where T : IGraphicalComponent

{

return Scale(comp.GetGraphicalElementBounds(), comp.GetGraphicalElementScale());

}

public static RectangleF? GetOffsetScaledGraphicalElementBounds<T>(this T comp) where T : IGraphicalComponent

{

return Offset(Scale(comp.GetGraphicalElementBounds(), comp.GetGraphicalElementScale()), comp.GetGraphicalElementLocation());

}

public static IComponent Copy<T>(this T comp) where T : IComponent

{

IComponent copy = ReadWriteImplementation.Constructors[comp.GetComponentID()](comp.GetComponentDetails());

if (comp is IGraphicalComponent graphicalComp && copy is IGraphicalComponent graphicalCopy)

{

graphicalCopy.SetGraphicalElementScale(graphicalComp.GetGraphicalElementScale());

graphicalCopy.SetGraphicalElementLocation(graphicalComp.GetGraphicalElementLocation());

}

if (comp is IBoardInterfaceComponent interfaceComp && copy is IBoardInterfaceComponent interfaceCopy)

{

interfaceCopy.SetInterfaceLocation(interfaceComp.GetInterfaceLocation());

}

if (comp is IBoardContainerComponent contComp && copy is IBoardContainerComponent contCopy)

{

Board intBoard = contComp.GetInternalBoard();

if (!(intBoard is null))

{

contCopy.ProvideInternalBoard(intBoard.Copy());

}

}

return copy;

}

}

### Pin

public class Pin

{

private State CurrentState, OriginalState;

public State GetStateForComponent()

{

return OriginalState.Pulled();

}

public State GetStateForWireComponent()

{

return CurrentState;

}

public State GetStateForWire()

{

return CurrentState;

}

public State GetStateForDisplay()

{

return CurrentState;

}

public void SetState(State state)

{

CurrentState = CurrentState.WireJoin(state);

}

public void ResetToFloating()

{

CurrentState = State.FLOATING;

OriginalState = State.FLOATING;

}

public void SetupForTick()

{

OriginalState = CurrentState;

CurrentState = Pin.State.FLOATING;

}

}

#### State

public enum State

{

FLOATING, LOW, PULLEDLOW, HIGH, PULLEDHIGH, ILLEGAL

}

### StateExtensions

static class StateExtensions

{

private static Dictionary<Pin.State, Pin.State> NotOpTable = new Dictionary<Pin.State, Pin.State>

{

{ Pin.State.FLOATING, Pin.State.FLOATING },

{ Pin.State.LOW, Pin.State.HIGH },

{ Pin.State.HIGH, Pin.State.LOW },

{ Pin.State.ILLEGAL, Pin.State.ILLEGAL }

};

private static Dictionary<Pin.State, Pin.State> PullTable = new Dictionary<Pin.State, Pin.State>

{

{ Pin.State.FLOATING, Pin.State.FLOATING },

{ Pin.State.LOW, Pin.State.LOW },

{ Pin.State.PULLEDLOW, Pin.State.LOW },

{ Pin.State.HIGH, Pin.State.HIGH },

{ Pin.State.PULLEDHIGH, Pin.State.HIGH },

{ Pin.State.ILLEGAL, Pin.State.ILLEGAL }

};

private static Dictionary<(Pin.State state1, Pin.State state2), Pin.State> GenericOpTable = new Dictionary<(Pin.State state1, Pin.State state2), Pin.State>

{

{ (Pin.State.FLOATING, Pin.State.FLOATING), Pin.State.FLOATING },

{ (Pin.State.FLOATING, Pin.State.LOW), Pin.State.LOW },

{ (Pin.State.FLOATING, Pin.State.HIGH), Pin.State.HIGH },

{ (Pin.State.FLOATING, Pin.State.ILLEGAL), Pin.State.ILLEGAL },

{ (Pin.State.ILLEGAL, Pin.State.LOW), Pin.State.ILLEGAL },

{ (Pin.State.ILLEGAL, Pin.State.HIGH), Pin.State.ILLEGAL },

{ (Pin.State.ILLEGAL, Pin.State.ILLEGAL), Pin.State.ILLEGAL }

};

private static BinOpTable AndOpTable = new BinOpTable(new Dictionary<(Pin.State state1, Pin.State state2), Pin.State>

{

{ (Pin.State.LOW, Pin.State.LOW), Pin.State.LOW },

{ (Pin.State.LOW, Pin.State.HIGH), Pin.State.LOW },

{ (Pin.State.HIGH, Pin.State.HIGH), Pin.State.HIGH }

}.Concat(GenericOpTable));

private static BinOpTable OrOpTable = new BinOpTable(new Dictionary<(Pin.State state1, Pin.State state2), Pin.State>

{

{ (Pin.State.LOW, Pin.State.LOW), Pin.State.LOW },

{ (Pin.State.LOW, Pin.State.HIGH), Pin.State.HIGH },

{ (Pin.State.HIGH, Pin.State.HIGH), Pin.State.HIGH }

}.Concat(GenericOpTable));

private static BinOpTable XorOpTable = new BinOpTable(new Dictionary<(Pin.State state1, Pin.State state2), Pin.State>

{

{ (Pin.State.LOW, Pin.State.LOW), Pin.State.LOW },

{ (Pin.State.LOW, Pin.State.HIGH), Pin.State.HIGH },

{ (Pin.State.HIGH, Pin.State.HIGH), Pin.State.LOW }

}.Concat(GenericOpTable));

private static BinOpTable WireJoinTable = new BinOpTable

{

{ (Pin.State.FLOATING, Pin.State.FLOATING), Pin.State.FLOATING },

{ (Pin.State.FLOATING, Pin.State.LOW), Pin.State.LOW },

{ (Pin.State.FLOATING, Pin.State.PULLEDLOW), Pin.State.PULLEDLOW },

{ (Pin.State.FLOATING, Pin.State.HIGH), Pin.State.HIGH },

{ (Pin.State.FLOATING, Pin.State.PULLEDHIGH), Pin.State.PULLEDHIGH },

{ (Pin.State.FLOATING, Pin.State.ILLEGAL), Pin.State.ILLEGAL },

{ (Pin.State.LOW, Pin.State.LOW), Pin.State.LOW },

{ (Pin.State.LOW, Pin.State.PULLEDLOW), Pin.State.LOW },

{ (Pin.State.LOW, Pin.State.HIGH), Pin.State.ILLEGAL },

{ (Pin.State.LOW, Pin.State.PULLEDHIGH), Pin.State.LOW },

{ (Pin.State.LOW, Pin.State.ILLEGAL), Pin.State.ILLEGAL },

{ (Pin.State.PULLEDLOW, Pin.State.PULLEDLOW), Pin.State.PULLEDLOW },

{ (Pin.State.PULLEDLOW, Pin.State.HIGH), Pin.State.HIGH },

{ (Pin.State.PULLEDLOW, Pin.State.PULLEDHIGH), Pin.State.ILLEGAL },

{ (Pin.State.PULLEDLOW, Pin.State.ILLEGAL), Pin.State.ILLEGAL },

{ (Pin.State.HIGH, Pin.State.HIGH), Pin.State.HIGH },

{ (Pin.State.HIGH, Pin.State.PULLEDHIGH), Pin.State.HIGH },

{ (Pin.State.HIGH, Pin.State.ILLEGAL), Pin.State.ILLEGAL },

{ (Pin.State.PULLEDHIGH, Pin.State.PULLEDHIGH), Pin.State.PULLEDHIGH },

{ (Pin.State.PULLEDHIGH, Pin.State.ILLEGAL), Pin.State.ILLEGAL },

{ (Pin.State.ILLEGAL, Pin.State.ILLEGAL), Pin.State.ILLEGAL }

};

public static Pin.State WireJoin(this Pin.State state1, Pin.State state2)

{

return WireJoinTable[state1, state2];

}

public static Pin.State Not(this Pin.State state)

{

return NotOpTable[state];

}

public static Pin.State And(this Pin.State state1, Pin.State state2)

{

return AndOpTable[state1, state2];

}

public static Pin.State Or(this Pin.State state1, Pin.State state2)

{

return OrOpTable[state1, state2];

}

public static Pin.State Xor(this Pin.State state1, Pin.State state2)

{

return XorOpTable[state1, state2];

}

public static Pin.State Pulled(this Pin.State state)

{

return PullTable[state];

}

}

#### BinOpTable

private class BinOpTable : Dictionary<(Pin.State state1, Pin.State state2), Pin.State>

{

private static (Pin.State state1, Pin.State state2) simplifyStates((Pin.State state1, Pin.State state2) states)

{

if (states.state1 > states.state2)

{

return (states.state2, states.state1);

}

return states;

}

public new Pin.State this[(Pin.State state1, Pin.State state2) states]

{

get => base[simplifyStates(states)];

set => base[simplifyStates(states)] = value;

}

public Pin.State this[Pin.State state1, Pin.State state2]

{

get => base[simplifyStates((state1, state2))];

set => base[simplifyStates((state1, state2))] = value;

}

public BinOpTable(IEnumerable<KeyValuePair<(Pin.State state1, Pin.State state2), Pin.State>> kvps) : base(kvps.ToDictionary(kvp => simplifyStates(kvp.Key), kvp => kvp.Value)) { }

public BinOpTable(IDictionary<(Pin.State state1, Pin.State state2), Pin.State> dict) : base(dict.ToDictionary(kvp => simplifyStates(kvp.Key), kvp => kvp.Value)) { }

public BinOpTable() : base() { }

public new void Add((Pin.State state1, Pin.State state2) key, Pin.State val)

{

base.Add(simplifyStates(key), val);

}

}

### SideEnumExtensions

public static class SideEnumExtensions

{

public static bool IsLeftRight(this Board.InterfaceLocation.SideEnum side)

{

return ((byte)side & 0b010) != 0;

}

public static bool IsBottomRight(this Board.InterfaceLocation.SideEnum side)

{

return ((byte)side & 0b001) != 0;

}

public static bool IsTop(this Board.InterfaceLocation.SideEnum side)

{

return !side.IsLeftRight() && !side.IsBottomRight();

}

public static bool IsBottom(this Board.InterfaceLocation.SideEnum side)

{

return !side.IsLeftRight() && side.IsBottomRight();

}

public static bool IsLeft(this Board.InterfaceLocation.SideEnum side)

{

return side.IsLeftRight() && !side.IsBottomRight();

}

public static bool IsRight(this Board.InterfaceLocation.SideEnum side)

{

return side.IsLeftRight() && side.IsBottomRight();

}

public static Board.InterfaceLocation.SideEnum ToggleLeftRight(this Board.InterfaceLocation.SideEnum side)

{

return (Board.InterfaceLocation.SideEnum)((byte)side ^ 0b10);

}

public static Board.InterfaceLocation.SideEnum ToggleBottomRight(this Board.InterfaceLocation.SideEnum side)

{

return (Board.InterfaceLocation.SideEnum)((byte)side ^ 0b01);

}

public static Board.InterfaceLocation.SideEnum ToggleLeftRightIf(this Board.InterfaceLocation.SideEnum side, bool cond)

{

return cond ? side.ToggleLeftRight() : side;

}

public static Board.InterfaceLocation.SideEnum ToggleBottomRightIf(this Board.InterfaceLocation.SideEnum side, bool cond)

{

return cond ? side.ToggleBottomRight() : side;

}

public static Board.InterfaceLocation.SideEnum WithLeftRightAs(this Board.InterfaceLocation.SideEnum side, bool cond)

{

return (Board.InterfaceLocation.SideEnum)((cond ? 0b10 : 0b00) + ((byte)side & 0b01));

}

public static Board.InterfaceLocation.SideEnum WithBottomRightAs(this Board.InterfaceLocation.SideEnum side, bool cond)

{

return (Board.InterfaceLocation.SideEnum)((cond ? 0b01 : 0b00) + ((byte)side & 0b10));

}

}

### Board

public class Board : InstanceTracker<Board>.ITrackable

{

public static InstanceTracker<Board> AllBoards = new InstanceTracker<Board>();

private uint trackingID;

public void SetTrackingID(uint id)

{

trackingID = id;

}

public uint GetTrackingID()

{

return trackingID;

}

private DefaultDictionary<Pos, Pin> Pins;

private HashSet<Wire> Wires = new HashSet<Wire>();

private HashSet<Wire> AllWires = new HashSet<Wire>();

private DefaultDictionary<Pos, int> ConnectionsToPin = new DefaultDictionary<Pos, int>();

private HashSet<IComponent> Components = new HashSet<IComponent>();

private HashSet<IWireComponent> WireComponents = new HashSet<IWireComponent>();

private HashSet<IWireComponent> AllWireComponents = new HashSet<IWireComponent>();

private HashSet<IComponent> NonWireComponents = new HashSet<IComponent>();

private HashSet<IBoardInterfaceComponent> InterfaceComponents = new HashSet<IBoardInterfaceComponent>();

private HashSet<IBoardInputComponent> InputComponents = new HashSet<IBoardInputComponent>();

private HashSet<IBoardOutputComponent> OutputComponents = new HashSet<IBoardOutputComponent>();

private List<IGraphicalComponent> GraphicalComponents = new List<IGraphicalComponent>();

private HashSet<IBoardContainerComponent> ContainerComponents = new HashSet<IBoardContainerComponent>();

private Board owner;

public void SetOwnerBoard(Board owner)

{

this.owner = owner;

}

public void ResetOwnerBoard()

{

owner = null;

}

public Board GetTopLevelBoard()

{

if (owner is null)

{

return this;

}

return owner.GetTopLevelBoard();

}

private Size? externalSize;

public Size ExternalSize

{

get

{

if (!externalSize.HasValue)

{

int bidirCount = InputComponents.Count() + OutputComponents.Count() - InterfaceComponents.Count();

int vertLimit = Math.Max(Math.Max(InputComponents.Count(), OutputComponents.Count()) - bidirCount, 1),

horiLimit = Math.Max(Math.Max(bidirCount, vertLimit / 2), 1);

externalSize = new Size(vertLimit \* 2, horiLimit \* 2);

SizeChanged?.Invoke();

}

return externalSize.Value;

}

set

{

externalSize = value;

SizeChanged?.Invoke();

}

}

public event Action SizeChanged;

public string Name;

public Board(string name, Size? externalSize = null)

{

Name = name;

if (externalSize.HasValue)

{

ExternalSize = externalSize.Value;

}

Pins = new DefaultDictionary<Pos, Pin>(() => new Pin(), (pos) => ConnectionsToPin[pos] != 0);

AllBoards.Add(this);

}

public void AddWire(Wire wire)

{

Wires.Add(wire);

AllWires.Add(wire);

ConnectionsToPin[wire.Pos1]++;

ConnectionsToPin[wire.Pos2]++;

}

public void RemoveWire(Wire wire)

{

Wires.Remove(wire);

AllWires.Remove(wire);

ConnectionsToPin[wire.Pos1]--;

ConnectionsToPin[wire.Pos2]--;

}

public Wire[] GetAllWires()

{

return Wires.ToArray();

}

public IComponent[] GetComponents()

{

return Components.ToArray();

}

public IWireComponent[] GetWireComponents()

{

return WireComponents.ToArray();

}

public IComponent[] GetNonWireComponents()

{

return NonWireComponents.ToArray();

}

public IBoardInterfaceComponent[] GetInterfaceComponents()

{

return InterfaceComponents.ToArray();

}

public IBoardInterfaceComponent GetInterfaceComponent(string name)

{

foreach (IBoardInterfaceComponent comp in InterfaceComponents)

{

if (comp.GetComponentName() == name)

{

return comp;

}

}

return null;

}

public IGraphicalComponent[] GetGraphicalComponents()

{

return GraphicalComponents.ToArray();

}

public IGraphicalComponent GetGraphicalComponent(int index)

{

return GraphicalComponents[index];

}

public IBoardInputComponent[] GetInputComponents()

{

return InputComponents.ToArray();

}

public IBoardInputComponent GetInputComponent(string name)

{

foreach (IBoardInputComponent comp in InputComponents)

{

if (comp.GetComponentName() == name)

{

return comp;

}

}

return null;

}

public IBoardOutputComponent[] GetOutputComponents()

{

return OutputComponents.ToArray();

}

public IBoardOutputComponent GetOutputComponent(string name)

{

foreach (IBoardOutputComponent comp in OutputComponents)

{

if (comp.GetComponentName() == name)

{

return comp;

}

}

return null;

}

public IBoardContainerComponent[] GetContainerComponents()

{

return ContainerComponents.ToArray();

}

public void SupplyInternalBoards(Board[] internalBoards)

{

Dictionary<string, Board> internalBoardsFromName = internalBoards.ToDictionary(board => board.Name);

Queue<Board> unsuppliedBoards = new Queue<Board>(); // Code Reference: Queue usage

unsuppliedBoards.Enqueue(this);

string boardName;

Board supplyingBoard, providedBoard;

while (unsuppliedBoards.Count > 0)

{

supplyingBoard = unsuppliedBoards.Dequeue();

foreach (IBoardContainerComponent contComp in supplyingBoard.ContainerComponents) // for every ContainerComponent, there is a new board needed

{

boardName = contComp.GetInternalBoardName(); // so we find out what type it is

if (internalBoardsFromName.ContainsKey(boardName)) // and if we have that type

{

providedBoard = internalBoardsFromName[boardName].Copy(supplyBoards: false); // we create one

unsuppliedBoards.Enqueue(providedBoard); // add it to the queue

contComp.ProvideInternalBoard(providedBoard); // and also give a reference to the ContainerComponent that wants it

} else // and if we don't have that type

{

throw new InvalidDataException($"{supplyingBoard.Name} board contains {boardName} board, which has not been supplied"); // throw an exception

}

}

}

}

public Board[] GetBoardList()

{

List<Board> checkedBoardList = new List<Board>(), uncheckedBoardList = new List<Board> { this };

// Code Reference: List operations

Func<string, bool> notSeen = boardName => !checkedBoardList.Select(checkedBoard => checkedBoard.Name).Concat(uncheckedBoardList.Select(uncheckedBoard => uncheckedBoard.Name)).Contains(boardName);

while (uncheckedBoardList.Count > 0)

{

foreach (IBoardContainerComponent contComp in uncheckedBoardList[0].GetContainerComponents())

{

if (notSeen(contComp.GetInternalBoardName()))

{

Board intBoard = contComp.GetInternalBoard();

if (!(intBoard is null))

{

uncheckedBoardList.Add(intBoard);

}

}

}

checkedBoardList.Add(uncheckedBoardList[0]);

uncheckedBoardList.RemoveAt(0);

}

return checkedBoardList.ToArray();

}

public void TickSetup()

{

ClearUnusedPins();

foreach (Pin pin in Pins.Values)

{

pin.SetupForTick();

}

foreach (IBoardContainerComponent boardContainerComp in ContainerComponents)

{

boardContainerComp.GetInternalBoard().TickSetup();

}

}

public void TickComponents()

{

foreach (IComponent nonWireComp in NonWireComponents)

{

nonWireComp.Tick();

}

}

private bool SubTickWire(Wire wire)

{

Pin pin1 = wire.Pin1, pin2 = wire.Pin2;

Pin.State state1 = pin1.GetStateForWire(), state2 = pin2.GetStateForWire();

if (state1 != state2)

{

pin1.SetState(state2);

pin2.SetState(state1);

return true;

}

return false;

}

public bool SubTickWires()

{

bool retVal = false;

foreach (Wire wire in AllWires)

{

retVal |= SubTickWire(wire);

}

foreach (IWireComponent wireComp in AllWireComponents)

{

wireComp.Tick();

retVal |= wireComp.TickAgain();

}

return retVal;

}

public void TickWires()

{

while (SubTickWires()) { }

}

public void Tick()

{

TickSetup();

TickComponents();

TickWires();

}

public Pin this[Pos pos] => Pins[pos];

private string GuaranteeUniqueName(string current, string[] existing)

{

string baseName = current.TrimEnd("0123456789".ToArray());

string baseNumberString = current.Substring(baseName.Length);

int baseNumber = 0;

int.TryParse(baseNumberString, out baseNumber);

while (existing.Contains(current))

{

baseNumber++;

current = baseName + baseNumber.ToString();

}

return current;

}

private void CountComponentPins(IComponent comp)

{

foreach (Pos pos in comp.GetAllUniquePinPositions())

{

ConnectionsToPin[pos]++;

}

}

private void AddContainedBoardWires(IBoardContainerComponent comp)

{

AllWires.UnionWith(comp.GetInternalBoard().AllWires);

AllWireComponents.UnionWith(comp.GetInternalBoard().AllWireComponents);

if (!(owner is null))

{

owner.AddContainedBoardWires(comp);

}

}

internal void AddComponent(IComponent comp)

{

foreach (IComponent otherComp in Components)

{

if (otherComp.GetOffsetComponentBounds().IntersectsWith(comp.GetOffsetComponentBounds()))

{

throw new PlacementException("cant place on another component. this error shouldn't be in the final product");

}

}

Components.Add(comp);

if (comp is IGraphicalComponent graphicalComp)

{

GraphicalComponents.Add(graphicalComp);

}

if (comp is IWireComponent wireComp)

{

WireComponents.Add(wireComp);

AllWireComponents.Add(wireComp);

} else

{

NonWireComponents.Add(comp);

}

if (comp is IBoardContainerComponent contComp)

{

ContainerComponents.Add(contComp);

contComp.PromiseDetails(CountComponentPins);

contComp.PromiseDetails(AddContainedBoardWires);

} else

{

CountComponentPins(comp);

}

if (comp is IBoardInterfaceComponent interfaceComp)

{

InterfaceLocation interfaceLoc = interfaceComp.GetInterfaceLocation();

if (interfaceLoc.Distance == 0)

{

int[] existingLocs = GetInterfaceComponents().Where(thisComp => thisComp.GetInterfaceLocation().Side == interfaceLoc.Side).Select(thisComp => thisComp.GetInterfaceLocation().Distance).ToArray();

for (int newDist = 1; true; newDist += 2)

{

if (!existingLocs.Contains(newDist))

{

interfaceLoc.Distance = newDist;

interfaceComp.SetInterfaceLocation(interfaceLoc);

break;

}

}

}

interfaceComp.SetComponentName(GuaranteeUniqueName(interfaceComp.GetComponentName(),

InputComponents.Select(thisComp => thisComp.GetComponentName()).Concat(OutputComponents.Select(thisComp => thisComp.GetComponentName())).ToArray()));

InterfaceComponents.Add(interfaceComp);

if (comp is IBoardInputComponent inpComp)

{

InputComponents.Add(inpComp);

}

if (comp is IBoardOutputComponent outpComp)

{

OutputComponents.Add(outpComp);

}

bool isSide = interfaceLoc.Side.IsLeftRight();

while (interfaceLoc.Distance >= (isSide ? ExternalSize.Height : ExternalSize.Width))

{

ExternalSize = new Size(ExternalSize.Width + (isSide ? 0 : 1), ExternalSize.Height + (isSide ? 1 : 0));

}

}

}

internal void RemoveComponent(IComponent comp)

{

comp.GetAllUniquePinPositions().Select(pos => ConnectionsToPin[pos]--);

Components.Remove(comp);

if (comp is IGraphicalComponent graphicalComp)

{

GraphicalComponents.Remove(graphicalComp);

}

if (comp is IWireComponent wireComp)

{

WireComponents.Remove(wireComp);

AllWireComponents.Remove(wireComp);

} else

{

NonWireComponents.Remove(comp);

}

if (comp is IBoardInterfaceComponent interfaceComp)

{

InterfaceComponents.Remove(interfaceComp);

}

if (comp is IBoardInputComponent inpComp)

{

InputComponents.Remove(inpComp);

}

if (comp is IBoardOutputComponent outpComp)

{

OutputComponents.Remove(outpComp);

}

if (comp is IBoardContainerComponent contComp)

{

ContainerComponents.Remove(contComp);

AllWires.ExceptWith(contComp.GetInternalBoard().AllWires);

AllWireComponents.ExceptWith(contComp.GetInternalBoard().AllWireComponents);

}

}

protected void ClearUnusedPins()

{

HashSet<Pos> keepPinPositions = new HashSet<Pos>();

foreach (IComponent comp in Components)

{

keepPinPositions.UnionWith(comp.GetAllPinPositions());

}

foreach (Wire wire in Wires)

{

keepPinPositions.Add(wire.Pos1);

keepPinPositions.Add(wire.Pos2);

}

HashSet<Pos> removePinPositions = new HashSet<Pos>();

removePinPositions.UnionWith(Pins.Keys);

removePinPositions.ExceptWith(keepPinPositions);

foreach (Pos pinPos in removePinPositions)

{

Pins.Remove(pinPos);

}

}

public void Render(Graphics graphics, bool simulating, Rectangle bounds, ColourScheme colourScheme)

{

Pen gridPen = new Pen(colourScheme.Grid, 0.005F);

for (int x = bounds.Left; x <= bounds.Right; x++)

{

graphics.DrawLine(gridPen, x, bounds.Top, x, bounds.Bottom);

}

for (int y = bounds.Top; y <= bounds.Bottom; y++)

{

graphics.DrawLine(gridPen, bounds.Left, y, bounds.Right, y);

}

Pin pin;

foreach (Pos pinPos in Pins.Keys)

{

if (bounds.Contains(pinPos.X, pinPos.Y))

{

pin = Pins[pinPos];

if (ConnectionsToPin[pinPos] == 0)

{

continue;

}

if (ConnectionsToPin[pinPos] != 2)

{

graphics.FillEllipse(new SolidBrush(colourScheme.GetWireColour(pin.GetStateForDisplay())), pinPos.X - 0.05F, pinPos.Y - 0.05F, 0.1F, 0.1F);

}

if (simulating)

{

graphics.DrawString(pin.GetStateForDisplay().ToString(), new Font("arial", 0.1F), Brushes.Black, pinPos.X, pinPos.Y);

}

}

}

graphics.DrawEllipse(new Pen(colourScheme.Grid, 0.01F), -0.1F, -0.1F, 0.2F, 0.2F);

Matrix matrix;

foreach (IComponent comp in Components)

{

if (comp.GetOffsetComponentBounds().IntersectsWith(bounds))

{

using (new TransformRestorer(graphics))

{

matrix = comp.GetRenderMatrix();

graphics.MultiplyTransform(matrix);

comp.Render(graphics, simulating, colourScheme);

if (comp is IGraphicalComponent graphicalComp)

{

graphicalComp.RenderGraphicalElement(graphics, simulating, colourScheme);

}

RectangleF compBounds = comp.GetComponentBounds();

}

}

}

foreach (Wire wire in Wires)

{

graphics.DrawLine(new Pen(simulating ? colourScheme.GetWireColour(wire.Pin1.GetStateForDisplay()) : colourScheme.Wire, 0.01F), new Point(wire.Pos1.X, wire.Pos1.Y), new Point(wire.Pos2.X, wire.Pos2.Y));

}

}

public bool CheckAllowed(RectangleF bounds)

{

foreach (IComponent comp in Components)

{

if (comp.GetOffsetComponentBounds().IntersectsWith(bounds))

{

return false;

}

}

return true;

}

public void ResetToFloating()

{

foreach (Pin pin in Pins.Values)

{

pin.ResetToFloating();

}

}

public void ResetForSimulation()

{

ResetToFloating();

foreach (IComponent comp in Components)

{

comp.ResetToDefault();

}

}

private bool TrySimplifyWirePair(Wire wire1, Wire wire2)

{

Func<Pos, int> getCompOrd, getOtherOrd;

Func<int, int, Pos> backToPos;

int[] compOrd;

int min, max, otherOrd;

if (wire1.TrySplitWire(wire2.Pos1) ||

wire1.TrySplitWire(wire2.Pos2) ||

wire2.TrySplitWire(wire1.Pos1) ||

wire2.TrySplitWire(wire1.Pos2))

{

return true;

}

if ((ConnectionsToPin[wire1.Pos1] == 2 && (wire1.Pos1 == wire2.Pos1 || wire1.Pos1 == wire2.Pos2)) ||

(ConnectionsToPin[wire1.Pos2] == 2 && (wire1.Pos2 == wire2.Pos1 || wire1.Pos2 == wire2.Pos2)))

{

if (wire1.IsHori() && wire2.IsHori())

{

getCompOrd = pos => pos.Y;

getOtherOrd = pos => pos.X;

backToPos = (comp, other) => new Pos(other, comp);

}

else if (wire1.IsHori() && wire2.IsHori())

{

getCompOrd = pos => pos.X;

getOtherOrd = pos => pos.Y;

backToPos = (comp, other) => new Pos(comp, other);

}

else

{

return false;

}

compOrd = new int[] { getCompOrd(wire1.Pos1), getCompOrd(wire1.Pos2), getCompOrd(wire2.Pos1), getCompOrd(wire2.Pos2) };

otherOrd = getOtherOrd(wire1.Pos1);

min = compOrd.Aggregate(Math.Min);

max = compOrd.Aggregate(Math.Max);

new Wire(backToPos(min, otherOrd), backToPos(max, otherOrd), this);

wire1.Remove();

wire2.Remove();

return true;

}

return false;

}

public void SimplifyWires()

{

HashSet<Wire> removeWires = new HashSet<Wire>();

Wire[] wires;

HashSet<Wire> addWires = new HashSet<Wire>();

bool changeMade;

do

{

changeMade = false;

foreach (Wire wire in Wires)

{

if (wire.Pos1 == wire.Pos2)

{

removeWires.Add(wire);

}

}

foreach (Wire removeWire in removeWires)

{

removeWire.Remove();

}

removeWires.Clear();

wires = new Wire[Wires.Count];

Wires.CopyTo(wires);

for (int i = 0; i < wires.Length - 1; i++)

{

for (int j = i + 1; j < wires.Length; j++)

{

changeMade = TrySimplifyWirePair(wires[i], wires[j]);

if (changeMade)

{

break;

}

}

if (changeMade)

{

break;

}

}

} while (changeMade);

foreach (IBoardContainerComponent boardContainerComp in ContainerComponents)

{

boardContainerComp.GetInternalBoard().SimplifyWires();

}

}

public Dictionary<Pos, Pin.State> GetStateToCheckForChanges()

{

return Pins.ToDictionary(kvp => kvp.Key, kvp => kvp.Value.GetStateForDisplay());

}

public override string ToString()

{

return $"{Name} ({trackingID})";

}

private static string GetFilename(string boardname)

{

return $"Boards/{boardname}.brd";

}

public void Save(string filename)

{

using (FileStream file = File.Open(filename, FileMode.Create))

{

using (BinaryWriter bw = new BinaryWriter(file))

{

bw.Write(this);

}

}

}

public static Board Load(string filename)

{

using (FileStream file = File.Open(filename, FileMode.Open))

{

using (BinaryReader br = new BinaryReader(file))

{

return br.ReadBoard();

}

}

}

private Board CopySingle(string copyName = null)

{

Board copy = new Board(copyName ?? Name, new Size(ExternalSize.Width, ExternalSize.Height));

foreach (IComponent comp in Components)

{

comp.Copy().Place(comp.GetComponentPos(), comp.GetComponentRotation(), copy);

}

foreach (Wire wire in Wires)

{

new Wire(wire.Pos1, wire.Pos2, copy);

}

return copy;

}

public Board Copy(string copyName = null, bool supplyBoards = true)

{

Board copy = CopySingle(copyName);

if (supplyBoards)

{

Board[] copiedBoards = GetBoardList();

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

{

copiedBoards[i] = copiedBoards[i].CopySingle();

}

copy.SupplyInternalBoards(copiedBoards);

}

return copy;

}

public override bool Equals(object obj)

{

if (obj is Board board)

{

return this == board;

}

return false;

}

public override int GetHashCode()

{

return GetTrackingID().GetHashCode();

}

public static bool operator ==(Board board1, Board board2)

{

if (board1 is null) { if (board2 is null) { return true; } else { return false; } } else if (board2 is null) { return false; }

return board1.GetTrackingID() == board2.GetTrackingID();

}

public static bool operator !=(Board board1, Board board2)

{

if (board1 is null) { if (board2 is null) { return false; } else { return true; } } else if (board2 is null) { return true; }

return board1.GetTrackingID() != board2.GetTrackingID();

}

}

#### InterfaceLocation

public struct InterfaceLocation

{

public SideEnum Side;

public int Distance;

public InterfaceLocation(SideEnum side, int distance)

{

Side = side;

Distance = distance;

}

public static bool operator ==(InterfaceLocation loc1, InterfaceLocation loc2)

{

return loc1.Side == loc2.Side && loc1.Distance == loc2.Distance;

}

public static bool operator !=(InterfaceLocation loc1, InterfaceLocation loc2)

{

return loc1.Side != loc2.Side || loc1.Distance != loc2.Distance;

}

public override bool Equals(object other)

{

if (other is InterfaceLocation otherLoc)

{

return this == otherLoc;

}

return false;

}

public override int GetHashCode()

{

return (int)Side ^ Distance;

}

public override string ToString()

{

return $"({Side},{Distance})";

}

}

##### SideEnum

public enum SideEnum : byte {

Top = 0b00,

Bottom = 0b01,

Left = 0b10,

Right = 0b11,

}

## Components

### ComponentRegisterer

public static class ComponentRegisterer

{

public static void RegisterComponents()

{

ReadWriteImplementation.Constructors.Add(VarInpAndComponent.ID, VarInpAndComponent.Constructor);

ReadWriteImplementation.Constructors.Add(VarInpOrComponent.ID, VarInpOrComponent.Constructor);

ReadWriteImplementation.Constructors.Add(VarInpXorComponent.ID, VarInpXorComponent.Constructor);

ReadWriteImplementation.Constructors.Add(VarInpNandComponent.ID, VarInpNandComponent.Constructor);

ReadWriteImplementation.Constructors.Add(VarInpNorComponent.ID, VarInpNorComponent.Constructor);

ReadWriteImplementation.Constructors.Add(VarInpXnorComponent.ID, VarInpXnorComponent.Constructor);

ReadWriteImplementation.Constructors.Add(BufferComponent.ID, BufferComponent.Constructor);

ReadWriteImplementation.Constructors.Add(NotComponent.ID, NotComponent.Constructor);

ReadWriteImplementation.Constructors.Add(TristateBufferComponent.ID, TristateBufferComponent.Constructor);

ReadWriteImplementation.Constructors.Add(FixedStateComponent.ID, FixedStateComponent.Constructor);

ReadWriteImplementation.Constructors.Add(UserToggleInpComponent.ID, UserToggleInpComponent.Constructor);

ReadWriteImplementation.Constructors.Add(UserPulseInpComponent.ID, UserPulseInpComponent.Constructor);

ReadWriteImplementation.Constructors.Add(LogicProbeComponent.ID, LogicProbeComponent.Constructor);

ReadWriteImplementation.Constructors.Add(SevenSegmentComponent.ID, SevenSegmentComponent.Constructor);

ReadWriteImplementation.Constructors.Add(BoardInputComponent.ID, BoardInputComponent.Constructor);

ReadWriteImplementation.Constructors.Add(BoardOutputComponent.ID, BoardOutputComponent.Constructor);

ReadWriteImplementation.Constructors.Add(BoardBidirComponent.ID, BoardBidirComponent.Constructor);

ReadWriteImplementation.Constructors.Add(BoardContainerComponent.ID, BoardContainerComponent.Constructor);

ReadWriteImplementation.DefaultDetails.Add(VarInpAndComponent.ID, VarInpAndComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(VarInpOrComponent.ID, VarInpOrComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(VarInpXorComponent.ID, VarInpXorComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(VarInpNandComponent.ID, VarInpNandComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(VarInpNorComponent.ID, VarInpNorComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(VarInpXnorComponent.ID, VarInpXnorComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(BufferComponent.ID, BufferComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(NotComponent.ID, NotComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(TristateBufferComponent.ID, TristateBufferComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(FixedStateComponent.ID, FixedStateComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(UserToggleInpComponent.ID, UserToggleInpComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(UserPulseInpComponent.ID, UserPulseInpComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(LogicProbeComponent.ID, LogicProbeComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(SevenSegmentComponent.ID, SevenSegmentComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(BoardInputComponent.ID, BoardInputComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(BoardOutputComponent.ID, BoardOutputComponent.DefaultDetails);

ReadWriteImplementation.DefaultDetails.Add(BoardBidirComponent.ID, BoardBidirComponent.DefaultDetails);

}

}

### BaseComponent

abstract class BaseComponent : IComponent

{

private Pos ComponentPos;

private Rotation ComponentRotation;

private Board ComponentBoard;

public Pos GetComponentPos()

{

return ComponentPos;

}

public Rotation GetComponentRotation()

{

return ComponentRotation;

}

public Matrix GetRenderMatrix()

{

Matrix matrix = new Matrix();

matrix.Translate(ComponentPos.X, ComponentPos.Y);

matrix.Rotate((float)ComponentRotation);

return matrix;

}

public Board GetComponentBoard()

{

return ComponentBoard;

}

private bool isPlaced = false;

public bool IsPlaced() { return isPlaced; }

public void Place(Pos pos, Board board)

{

Place(pos, Rotation.ZERO, board);

}

public virtual void Place(Pos pos, Rotation rotation, Board board)

{

if (isPlaced)

{

throw new PlacementException("Already placed, can't place again.");

}

ComponentPos = pos;

ComponentRotation = rotation;

ComponentBoard = board;

board.AddComponent(this);

isPlaced = true;

}

public virtual void Remove() // can be removed if not placed

{

if (isPlaced)

{

ComponentBoard.RemoveComponent(this);

}

isPlaced = false;

}

public abstract void Tick();

public virtual void ResetToDefault() { }

public abstract Pos[] GetAllPinOffsets();

public abstract Pos[] GetAllPinPositions();

public abstract string GetComponentID();

public abstract string GetComponentDetails();

public abstract IComponent NonStaticConstructor(string details);

public abstract void Render(Graphics graphics, bool simulating, ColourScheme colourScheme);

public abstract void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme);

protected void DrawComponentFromPath(Graphics graphics, GraphicsPath path, ColourScheme colourScheme)

{

graphics.FillPath(new SolidBrush(colourScheme.ComponentBackground), path);

graphics.DrawPath(new Pen(colourScheme.ComponentEdge, 0.01F), path);

}

protected void DrawInversionCircle(Graphics graphics, PointF point, ColourScheme colourScheme, float rad = 0.2F)

{

graphics.FillEllipse(new SolidBrush(colourScheme.Background), point.X, point.Y - rad, 2 \* rad, 2 \* rad);

graphics.DrawEllipse(new Pen(colourScheme.ComponentEdge, 0.01F), point.X, point.Y - rad, 2 \* rad, 2 \* rad);

}

public abstract RectangleF GetComponentBounds();

protected RectangleF CreateSmallestRect(Pos[] containedPoints)

{

// Code Reference: List operations

return RectangleF.FromLTRB(

containedPoints.Select(offset => offset.X).Aggregate(Math.Min),

containedPoints.Select(offset => offset.Y).Aggregate(Math.Min),

containedPoints.Select(offset => offset.X).Aggregate(Math.Max),

containedPoints.Select(offset => offset.Y).Aggregate(Math.Max));

}

protected RectangleF GetDefaultComponentBounds()

{

Pos[] offsets = GetAllPinOffsets().Append(new Pos(0, 0)).ToArray();

return CreateSmallestRect(offsets);

}

public RectangleF GetOffsetComponentBounds()

{

RectangleF rect = GetComponentBounds();

Matrix matrix = GetRenderMatrix();

PointF[] corners = { new PointF(rect.Left, rect.Top), new PointF(rect.Right, rect.Bottom) };

matrix.TransformPoints(corners);

return RectangleF.FromLTRB(

Math.Min(corners[0].X, corners[1].X),

Math.Min(corners[0].Y, corners[1].Y),

Math.Max(corners[0].X, corners[1].X),

Math.Max(corners[0].Y, corners[1].Y)

).Round();

}

public override string ToString()

{

return $"{GetComponentID()}:{GetComponentDetails()}@{ComponentPos}";

}

private Pos[] RemoveDuplicates(Pos[] positions)

{

List<Pos> posList = new List<Pos>();

foreach (Pos pos in positions)

{

if (!posList.Contains(pos))

{

posList.Add(pos);

}

}

return posList.ToArray();

}

public Pos[] GetAllUniquePinPositions()

{

return RemoveDuplicates(GetAllPinPositions());

}

}

### InpOutpTools

abstract class InpOutpTools

{

public static Pos GetSingRotatedOffset(IComponent comp, Pos SingPinOffset) // returns offset, rotated

{

return SingPinOffset.Rotate(comp.GetComponentRotation());

}

public static Pos GetSingPosition(IComponent comp, Pos SingPinOffset) // returns pin position

{

return comp.GetComponentPos().Add(GetSingRotatedOffset(comp, SingPinOffset));

}

public static Pin GetSingPin(IComponent comp, Pos SingPinOffset)

{

return comp.GetComponentBoard()[GetSingPosition(comp, SingPinOffset)];

}

public static Pos[] GetMultRotatedOffsets(IComponent comp, Pos[] MultPinOffset)

{

return MultPinOffset.Select(pinOffset => GetSingRotatedOffset(comp, pinOffset)).ToArray();

}

public static Pos[] GetMultPositions(IComponent comp, Pos[] MultPinOffset)

{

return MultPinOffset.Select(pinOffset => GetSingPosition(comp, pinOffset)).ToArray();

}

public static Pin[] GetMultPins(IComponent comp, Pos[] MultPinOffset)

{

return MultPinOffset.Select(pinOffset => GetSingPin(comp, pinOffset)).ToArray();

}

public static Color GetWireColour(bool simulating, Pos offset, ColourScheme colourScheme, IComponent comp)

{

if (simulating && comp.IsPlaced())

{

return colourScheme.GetWireColour(comp.GetComponentBoard()[offset.Rotate(comp.GetComponentRotation()).Add(comp.GetComponentPos())].GetStateForDisplay());

}

return colourScheme.Wire;

}

public static void DrawInpOutpLine(Graphics graphics, bool simulating, Pos pinOffset, PointF otherOffset, ColourScheme colourScheme, IComponent comp)

{

graphics.DrawLine(

new Pen(GetWireColour(simulating, pinOffset, colourScheme, comp), 0.01F),

pinOffset.X, pinOffset.Y, otherOffset.X, otherOffset.Y);

}

public static void DrawInpLine(Graphics graphics, bool simulating, Pos inpOffset, ColourScheme colourScheme, IComponent comp)

{

DrawInpOutpLine(graphics, simulating, inpOffset, new PointF(inpOffset.X + 1.5F, inpOffset.Y), colourScheme, comp);

}

public static void DrawOutpLine(Graphics graphics, bool simulating, Pos outpOffset, ColourScheme colourScheme, IComponent comp)

{

DrawInpOutpLine(graphics, simulating, outpOffset, new PointF(outpOffset.X - 1.5F, outpOffset.Y), colourScheme, comp);

}

}

### ISingInpComponent

interface ISingInpComponent : IComponent

{

Pos GetInpOffset();

Pos GetInpPosition();

Pin GetInpPin();

}

### IMultInpComponent

interface IMultInpComponent : IComponent

{

Pos[] GetInpOffsets();

Pos[] GetInpPositions();

Pin[] GetInpPins();

}

### ISingOutpComponent

interface ISingOutpComponent : IComponent

{

Pos GetOutpOffset();

Pos GetOutpPosition();

Pin GetOutpPin();

}

### IMultInpComponent

interface IMultOutpComponent : IComponent

{

Pos[] GetOutpOffsets();

Pos[] GetOutpPositions();

Pin[] GetOutpPins();

}

### ISingInpSingOutpComponent

interface ISingInpSingOutpComponent : ISingInpComponent, ISingOutpComponent { }

### ISingInpMultOutpComponent

interface ISingInpMultOutpComponent : ISingInpComponent, IMultOutpComponent { }

### IMultInpSingOutpComponent

interface IMultInpSingOutpComponent : IMultInpComponent, ISingOutpComponent { }

### IMultInpMultOutpComponent

interface IMultInpMultOutpComponent : IMultInpComponent, IMultOutpComponent { }

### SingInpNoneOutpBaseComponent

abstract class SingInpNoneOutpBaseComponent : BaseComponent, ISingInpComponent

{

public abstract Pos GetInpOffset();

public Pos GetInpPosition()

{

return InpOutpTools.GetSingPosition(this, GetInpOffset());

}

public Pin GetInpPin()

{

return InpOutpTools.GetSingPin(this, GetInpOffset());

}

public override Pos[] GetAllPinOffsets()

{

return new Pos[] { GetInpOffset() };

}

public override Pos[] GetAllPinPositions()

{

return new Pos[] { GetInpPosition() };

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

InpOutpTools.DrawInpLine(graphics, simulating, GetInpOffset(), colourScheme, this);

RenderMainShape(graphics, simulating, colourScheme);

}

}

### MultInpNoneOutpBaseComponent

abstract class MultInpNoneOutpBaseComponent : BaseComponent, IMultInpComponent

{

public abstract Pos[] GetInpOffsets();

public Pos[] GetInpPositions()

{

return InpOutpTools.GetMultPositions(this, GetInpOffsets());

}

public Pin[] GetInpPins()

{

return InpOutpTools.GetMultPins(this, GetInpOffsets());

}

public override Pos[] GetAllPinOffsets()

{

return GetInpOffsets();

}

public override Pos[] GetAllPinPositions()

{

return GetInpPositions();

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

foreach (Pos inpOffset in GetInpOffsets())

{

InpOutpTools.DrawInpLine(graphics, simulating, inpOffset, colourScheme, this);

}

RenderMainShape(graphics, simulating, colourScheme);

}

}

### NoneInpSingOutpBaseComponent

abstract class NoneInpSingOutpBaseComponent : BaseComponent, ISingOutpComponent

{

public abstract Pos GetOutpOffset();

public Pos GetOutpPosition()

{

return InpOutpTools.GetSingPosition(this, GetOutpOffset());

}

public Pin GetOutpPin()

{

return InpOutpTools.GetSingPin(this, GetOutpOffset());

}

public override Pos[] GetAllPinOffsets()

{

return new Pos[] { GetOutpOffset() };

}

public override Pos[] GetAllPinPositions()

{

return new Pos[] { GetOutpPosition() };

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

InpOutpTools.DrawOutpLine(graphics, simulating, GetOutpOffset(), colourScheme, this);

RenderMainShape(graphics, simulating, colourScheme);

}

}

### NoneInpMultOutpBaseComponent

abstract class NoneInpMultOutpBaseComponent : BaseComponent, IMultOutpComponent

{

public abstract Pos[] GetOutpOffsets();

public Pos[] GetOutpPositions()

{

return InpOutpTools.GetMultPositions(this, GetOutpOffsets());

}

public Pin[] GetOutpPins()

{

return InpOutpTools.GetMultPins(this, GetOutpOffsets());

}

public override Pos[] GetAllPinOffsets()

{

return GetOutpOffsets();

}

public override Pos[] GetAllPinPositions()

{

return GetOutpPositions();

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

foreach (Pos outpOffset in GetOutpOffsets())

{

InpOutpTools.DrawOutpLine(graphics, simulating, outpOffset, colourScheme, this);

}

RenderMainShape(graphics, simulating, colourScheme);

}

}

### SingInpSingOutpBaseComponent

abstract class SingInpSingOutpBaseComponent : BaseComponent, ISingInpSingOutpComponent

{

public abstract Pos GetInpOffset();

public Pos GetInpPosition()

{

return InpOutpTools.GetSingPosition(this, GetInpOffset());

}

public Pin GetInpPin()

{

return InpOutpTools.GetSingPin(this, GetInpOffset());

}

public abstract Pos GetOutpOffset();

public Pos GetOutpPosition()

{

return InpOutpTools.GetSingPosition(this, GetOutpOffset());

}

public Pin GetOutpPin()

{

return InpOutpTools.GetSingPin(this, GetOutpOffset());

}

public override Pos[] GetAllPinOffsets()

{

return new Pos[] { GetInpOffset(), GetOutpOffset() };

}

public override Pos[] GetAllPinPositions()

{

return new Pos[] { GetInpPosition(), GetOutpPosition() };

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

InpOutpTools.DrawInpLine(graphics, simulating, GetInpOffset(), colourScheme, this);

InpOutpTools.DrawOutpLine(graphics, simulating, GetOutpOffset(), colourScheme, this);

RenderMainShape(graphics, simulating, colourScheme);

}

}

### SingInpMultOutpBaseComponent

abstract class SingInpMultOutpBaseComponent : BaseComponent, ISingInpMultOutpComponent

{

public abstract Pos GetInpOffset();

public Pos GetInpPosition()

{

return InpOutpTools.GetSingPosition(this, GetInpOffset());

}

public Pin GetInpPin()

{

return InpOutpTools.GetSingPin(this, GetInpOffset());

}

public abstract Pos[] GetOutpOffsets();

public Pos[] GetOutpPositions()

{

return InpOutpTools.GetMultPositions(this, GetOutpOffsets());

}

public Pin[] GetOutpPins()

{

return InpOutpTools.GetMultPins(this, GetOutpOffsets());

}

public override Pos[] GetAllPinOffsets()

{

return (new Pos[] { GetInpOffset() }).Concat(GetOutpOffsets()).ToArray();

}

public override Pos[] GetAllPinPositions()

{

return (new Pos[] { GetInpPosition() }).Concat(GetOutpPositions()).ToArray();

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

InpOutpTools.DrawInpLine(graphics, simulating, GetInpOffset(), colourScheme, this);

foreach (Pos outpOffset in GetOutpOffsets())

{

InpOutpTools.DrawOutpLine(graphics, simulating, outpOffset, colourScheme, this);

}

RenderMainShape(graphics, simulating, colourScheme);

}

}

### MultInpSingOutpBaseComponent

abstract class MultInpSingOutpBaseComponent : BaseComponent, IMultInpSingOutpComponent

{

public abstract Pos[] GetInpOffsets();

public Pos[] GetInpPositions()

{

return InpOutpTools.GetMultPositions(this, GetInpOffsets());

}

public Pin[] GetInpPins()

{

return InpOutpTools.GetMultPins(this, GetInpOffsets());

}

public abstract Pos GetOutpOffset();

public Pos GetOutpPosition()

{

return InpOutpTools.GetSingPosition(this, GetOutpOffset());

}

public Pin GetOutpPin()

{

return InpOutpTools.GetSingPin(this, GetOutpOffset());

}

public override Pos[] GetAllPinOffsets()

{

return GetInpOffsets().Append(GetOutpOffset()).ToArray();

}

public override Pos[] GetAllPinPositions()

{

return GetInpPositions().Append(GetOutpPosition()).ToArray();

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

foreach (Pos inpOffset in GetInpOffsets())

{

InpOutpTools.DrawInpLine(graphics, simulating, inpOffset, colourScheme, this);

}

InpOutpTools.DrawOutpLine(graphics, simulating, GetOutpOffset(), colourScheme, this);

RenderMainShape(graphics, simulating, colourScheme);

}

}

### MultInpMultOutpBaseComponent

abstract class MultInpMultOutpBaseComponent : BaseComponent, IMultInpMultOutpComponent

{

public abstract Pos[] GetInpOffsets();

public Pos[] GetInpPositions()

{

return InpOutpTools.GetMultPositions(this, GetInpOffsets());

}

public Pin[] GetInpPins()

{

return InpOutpTools.GetMultPins(this, GetInpOffsets());

}

public abstract Pos[] GetOutpOffsets();

public Pos[] GetOutpPositions()

{

return InpOutpTools.GetMultPositions(this, GetOutpOffsets());

}

public Pin[] GetOutpPins()

{

return InpOutpTools.GetMultPins(this, GetOutpOffsets());

}

public override Pos[] GetAllPinOffsets()

{

return GetInpOffsets().Concat(GetOutpOffsets()).ToArray();

}

public override Pos[] GetAllPinPositions()

{

return GetInpPositions().Concat(GetOutpPositions()).ToArray();

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

foreach (Pos inpOffset in GetInpOffsets())

{

InpOutpTools.DrawInpLine(graphics, simulating, inpOffset, colourScheme, this);

}

foreach (Pos outpOffset in GetOutpOffsets())

{

InpOutpTools.DrawOutpLine(graphics, simulating, outpOffset, colourScheme, this);

}

RenderMainShape(graphics, simulating, colourScheme);

}

}

### BaseBufferComponent

abstract class BaseBufferComponent : MultInpSingOutpBaseComponent

{

public override Pos GetOutpOffset()

{

return new Pos(1, 0);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[]

{

new PointF(-0.5F, -0.5F),

new PointF(0.5F, 0),

new PointF(-0.5F, 0.5F)

});

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(0, 0.5F);

return rect;

}

}

### BufferComponent

class BufferComponent : BaseBufferComponent

{

public override Pos[] GetInpOffsets()

{

return new Pos[] { new Pos(-1, 0) };

}

public BufferComponent() { }

public override void Tick()

{

GetOutpPin().SetState(GetInpPins()[0].GetStateForComponent());

}

public static string ID = "BUFFER";

public static string DefaultDetails = "";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

return "";

}

public static BufferComponent Constructor(string details)

{

return new BufferComponent();

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

}

### NotComponent

class NotComponent : BaseBufferComponent

{

public override Pos[] GetInpOffsets()

{

return new Pos[] { new Pos(-1, 0) };

}

public NotComponent() { }

public override void Tick()

{

GetOutpPin().SetState(GetInpPins()[0].GetStateForComponent().Not());

}

public static string ID = "NOT";

public static string DefaultDetails = "";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

return "";

}

public static NotComponent Constructor(string details)

{

return new NotComponent();

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

base.RenderMainShape(graphics, simulating, colourScheme);

DrawInversionCircle(graphics, new PointF(0.5F, 0), colourScheme, 0.1F);

}

}

### TristateBufferComponent

class TristateBufferComponent : BaseBufferComponent

{

public override Pos[] GetInpOffsets()

{

return new Pos[] {

new Pos(-1, 0),

new Pos(0, -1)

};

}

public TristateBufferComponent() { }

public override void Tick()

{

Pin.State activationState = GetInpPins()[1].GetStateForComponent();

if (activationState == Pin.State.HIGH)

{

GetOutpPin().SetState(GetInpPins()[0].GetStateForComponent());

}

else if (activationState == Pin.State.LOW || activationState == Pin.State.FLOATING)

{

GetOutpPin().SetState(Pin.State.FLOATING);

}

else

{

GetOutpPin().SetState(Pin.State.ILLEGAL);

}

}

public static string ID = "TRISTATE";

public static string DefaultDetails = "";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

return "";

}

public static TristateBufferComponent Constructor(string details)

{

return new TristateBufferComponent();

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Height += 0.5F;

return rect;

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

InpOutpTools.DrawInpLine(graphics, simulating, GetInpOffsets()[0], colourScheme, this);

InpOutpTools.DrawInpOutpLine(graphics, simulating, GetInpOffsets()[1], new PointF(0, 0), colourScheme, this);

InpOutpTools.DrawOutpLine(graphics, simulating, GetOutpOffset(), colourScheme, this);

RenderMainShape(graphics, simulating, colourScheme);

}

}

### BaseVarInpComponent

abstract class BaseVarInpComponent : MultInpSingOutpBaseComponent, ISettingsComponent

{

private int InpCount;

private Pos[] InpOffsets;

private Pos OutpOffset;

private string Details;

private PositiveIntSettingDescription inputSettingDesc;

public override Pos[] GetInpOffsets()

{

return InpOffsets;

}

public override Pos GetOutpOffset()

{

return OutpOffset;

}

private void DefineInpOffsets()

{

if (InpCount < 2)

{

throw new PlacementException("Can't have less than 2 inputs.");

}

InpOffsets = new Pos[InpCount];

for (int inpNum = 0; inpNum < InpCount; inpNum++)

{

InpOffsets[inpNum] = new Pos(-2, (2 \* inpNum) - InpCount + 1);

}

Details = $"{InpCount}";

}

public BaseVarInpComponent(int inpCount)

{

OutpOffset = new Pos(2, 0);

InpCount = inpCount;

DefineInpOffsets();

}

protected abstract bool IsNegated();

public override void Tick()

{

// Code Reference: List operations

Pin.State state = GetInpPins().Select(inpPin => inpPin.GetStateForComponent()).Aggregate(Accumulator);

state = IsNegated() ? state.Not() : state;

GetOutpPin().SetState(state);

}

protected abstract Pin.State Accumulator(Pin.State state1, Pin.State state2);

public override string GetComponentDetails()

{

return Details;

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(0, 0.5F);

return rect;

}

public ISettingDescription[] GetSettingDescriptions()

{

inputSettingDesc = new PositiveIntSettingDescription("How many inputs should this component have?", InpCount);

return new ISettingDescription[] { inputSettingDesc };

}

public void ApplySettings()

{

InpCount = inputSettingDesc.GetValue();

DefineInpOffsets();

}

protected static int ParseInpCount(string details)

{

int inpCount;

if (int.TryParse(details, out inpCount))

{

return inpCount;

}

throw new PlacementException("Did not successfully parse int.");

}

protected GraphicsPath AddAndShape(GraphicsPath path)

{

float vertDist = InpOffsets.Length - 0.5F;

path.AddBeziers(new PointF[] {

new PointF(-1, -vertDist),

new PointF(0, -vertDist),

new PointF(1, -vertDist),

new PointF(1, 0),

new PointF(1, vertDist),

new PointF(0, vertDist),

new PointF(-1, vertDist)

});

path.CloseFigure();

return path;

}

protected GraphicsPath AddOrShape(GraphicsPath path)

{

float vertDist = InpOffsets.Length - 0.5F;

path.AddBezier(

new PointF(-1.5F, -vertDist),

new PointF(-0.5F, -vertDist),

new PointF(0.5F, -vertDist),

new PointF(1, 0)

);

path.AddBezier(

new PointF(1, 0),

new PointF(0.5F, vertDist),

new PointF(-0.5F, vertDist),

new PointF(-1.5F, vertDist)

);

path.AddBezier(

new PointF(-1.5F, vertDist),

new PointF(-0.5F, 0.5F),

new PointF(-0.5F, -0.5F),

new PointF(-1.5F, -vertDist)

);

path.CloseFigure();

return path;

}

protected GraphicsPath AddXorShape(GraphicsPath path)

{

float vertDist = InpOffsets.Length - 0.5F;

AddOrShape(path);

path.AddBezier(

new PointF(-1.75F, vertDist),

new PointF(-0.75F, 0.5F),

new PointF(-0.75F, -0.5F),

new PointF(-1.75F, -vertDist)

);

path.AddBezier(

new PointF(-1.75F, -vertDist),

new PointF(-0.75F, -0.5F),

new PointF(-0.75F, 0.5F),

new PointF(-1.75F, vertDist)

);

return path;

}

protected void DrawAndComponent(Graphics graphics, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

AddAndShape(path);

DrawComponentFromPath(graphics, path, colourScheme);

}

protected void DrawOrComponent(Graphics graphics, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

AddOrShape(path);

DrawComponentFromPath(graphics, path, colourScheme);

}

protected void DrawXorComponent(Graphics graphics, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

AddXorShape(path);

DrawComponentFromPath(graphics, path, colourScheme);

}

}

### VarInpAndComponent

class VarInpAndComponent : BaseVarInpComponent

{

public VarInpAndComponent(int inpCount) : base(inpCount) { }

protected override Pin.State Accumulator(Pin.State state1, Pin.State state2)

{

return state1.And(state2);

}

public static string ID = "AND";

public static string DefaultDetails = "2";

public override string GetComponentID()

{

return ID;

}

protected override bool IsNegated()

{

return false;

}

public static IComponent Constructor(string details)

{

return new VarInpAndComponent(ParseInpCount(details));

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

DrawAndComponent(graphics, colourScheme);

}

}

### VarInpOrComponent

class VarInpOrComponent : BaseVarInpComponent

{

public VarInpOrComponent(int inpCount) : base(inpCount) { }

protected override Pin.State Accumulator(Pin.State state1, Pin.State state2)

{

return state1.Or(state2);

}

public static string ID = "OR";

public static string DefaultDetails = "2";

public override string GetComponentID()

{

return ID;

}

protected override bool IsNegated()

{

return false;

}

public static IComponent Constructor(string details)

{

return new VarInpOrComponent(ParseInpCount(details));

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

DrawOrComponent(graphics, colourScheme);

}

}

### VarInpXorComponent

class VarInpXorComponent : BaseVarInpComponent

{

public VarInpXorComponent(int inpCount) : base(inpCount) { }

protected override Pin.State Accumulator(Pin.State state1, Pin.State state2)

{

return state1.Xor(state2);

}

public static string ID = "XOR";

public static string DefaultDetails = "2";

public override string GetComponentID()

{

return ID;

}

protected override bool IsNegated()

{

return false;

}

public static IComponent Constructor(string details)

{

return new VarInpXorComponent(ParseInpCount(details));

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

DrawXorComponent(graphics, colourScheme);

}

}

### VarInpNandComponent

class VarInpNandComponent : BaseVarInpComponent

{

public VarInpNandComponent(int inpCount) : base(inpCount) { }

protected override Pin.State Accumulator(Pin.State state1, Pin.State state2)

{

return state1.And(state2);

}

public static string ID = "NAND";

public static string DefaultDetails = "2";

public override string GetComponentID()

{

return ID;

}

protected override bool IsNegated()

{

return true;

}

public static IComponent Constructor(string details)

{

return new VarInpNandComponent(ParseInpCount(details));

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

DrawAndComponent(graphics, colourScheme);

DrawInversionCircle(graphics, new PointF(1, 0), colourScheme);

}

}

### VarInpNorComponent

class VarInpNorComponent : BaseVarInpComponent

{

public VarInpNorComponent(int inpCount) : base(inpCount) { }

protected override Pin.State Accumulator(Pin.State state1, Pin.State state2)

{

return state1.Or(state2);

}

public static string ID = "NOR";

public static string DefaultDetails = "2";

public override string GetComponentID()

{

return ID;

}

protected override bool IsNegated()

{

return true;

}

public static IComponent Constructor(string details)

{

return new VarInpNorComponent(ParseInpCount(details));

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

DrawOrComponent(graphics, colourScheme);

DrawInversionCircle(graphics, new PointF(1, 0), colourScheme);

}

}

### VarInpXnorComponent

class VarInpXnorComponent : BaseVarInpComponent

{

public VarInpXnorComponent(int inpCount) : base(inpCount) { }

protected override Pin.State Accumulator(Pin.State state1, Pin.State state2)

{

return state1.Xor(state2);

}

public static string ID = "XNOR";

public static string DefaultDetails = "2";

public override string GetComponentID()

{

return ID;

}

protected override bool IsNegated()

{

return true;

}

public static IComponent Constructor(string details)

{

return new VarInpXnorComponent(ParseInpCount(details));

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

DrawXorComponent(graphics, colourScheme);

DrawInversionCircle(graphics, new PointF(1, 0), colourScheme);

}

}

### FixedStateComponent

class FixedStateComponent : NoneInpSingOutpBaseComponent, ISettingsComponent

{

protected Pin.State OutputState;

protected virtual string getOutputDescriptor()

{

return "output";

}

protected EnumSettingDescription<Pin.State> stateSettingDesc;

public override Pos GetOutpOffset()

{

return new Pos(2, 0);

}

public FixedStateComponent(Pin.State state)

{

OutputState = state;

}

public override void Tick()

{

GetOutpPin().SetState(OutputState);

}

public static string ID = "FIXED";

public static string DefaultDetails = $"{(int)Pin.State.LOW}";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

return $"{(int)OutputState}";

}

public static IComponent Constructor(string details)

{

int outputState;

if (int.TryParse(details, out outputState))

{

return new FixedStateComponent((Pin.State)outputState);

}

throw new PlacementException("Did not successfully parse int.");

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(0, 0.5F);

rect.X -= 0.5F;

rect.Width += 0.5F;

return rect;

}

public virtual ISettingDescription[] GetSettingDescriptions()

{

stateSettingDesc = new EnumSettingDescription<Pin.State>($"What is the {getOutputDescriptor()} state for this component?", OutputState);

return new ISettingDescription[] { stateSettingDesc };

}

public virtual void ApplySettings()

{

OutputState = stateSettingDesc.GetValue();

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[] { new PointF(-0.5F, -0.5F), new PointF(0.5F, -0.5F), new PointF(1, 0), new PointF(0.5F, 0.5F), new PointF(-0.5F, 0.5F) });

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

}

### UserToggleInpComponent

class UserToggleInpComponent : NoneInpSingOutpBaseComponent, ISettingsComponent, IInteractibleComponent

{

protected Pin.State OutputState, StartState, OtherState;

protected virtual string getOutputDescriptor()

{

return "output";

}

protected EnumSettingDescription<Pin.State> startStateSettingDesc;

protected EnumSettingDescription<Pin.State> otherStateSettingDesc;

public override Pos GetOutpOffset()

{

return new Pos(2, 0);

}

public UserToggleInpComponent(Pin.State startState, Pin.State otherState)

{

OutputState = startState;

StartState = startState;

OtherState = otherState;

}

public override void Tick()

{

GetOutpPin().SetState(OutputState);

}

public void Interact()

{

if (OutputState == StartState)

{

OutputState = OtherState;

}

else

{

OutputState = StartState;

}

}

public static string ID = "TOGGLE";

public static string DefaultDetails = $"{(int)Pin.State.LOW},{(int)Pin.State.HIGH}";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

return $"{(int)StartState},{(int)OtherState}";

}

public static IComponent Constructor(string details)

{

int startState, otherState;

string[] states = details.Split(',');

if (int.TryParse(states[0], out startState) && int.TryParse(states[1], out otherState))

{

return new UserToggleInpComponent((Pin.State)startState, (Pin.State)otherState);

}

throw new PlacementException("Did not successfully parse int.");

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(0, 0.5F);

rect.X -= 0.5F;

rect.Width += 0.5F;

return rect;

}

public virtual ISettingDescription[] GetSettingDescriptions()

{

startStateSettingDesc = new EnumSettingDescription<Pin.State>($"What is the starting {getOutputDescriptor()} state for this component?", StartState);

otherStateSettingDesc = new EnumSettingDescription<Pin.State>($"What is the other {getOutputDescriptor()} state for this component?", OtherState);

return new ISettingDescription[] { startStateSettingDesc, otherStateSettingDesc };

}

public virtual void ApplySettings()

{

StartState = startStateSettingDesc.GetValue();

OtherState = otherStateSettingDesc.GetValue();

}

public override void ResetToDefault()

{

OutputState = StartState;

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[] { new PointF(-0.5F, -0.5F), new PointF(0.5F, -0.5F), new PointF(1, 0), new PointF(0.5F, 0.5F), new PointF(-0.5F, 0.5F) });

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

}

### UserPulseInpComponent

class UserPulseInpComponent : NoneInpSingOutpBaseComponent, IInteractibleComponent, ISettingsComponent

{

public override Pos GetOutpOffset()

{

return new Pos(2, 0);

}

protected int PulseLength;

protected Pin.State NormalState, PulseState;

protected PositiveIntSettingDescription PulseLengthSettingDesc;

protected EnumSettingDescription<Pin.State> NormalStateSettingDesc, PulseStateSettingDesc;

public static string ID = "PULSE";

public static string DefaultDetails = $"{20},{(int)Pin.State.LOW},{(int)Pin.State.HIGH}";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

return $"{PulseLength},{(int)NormalState},{(int)PulseState}";

}

public UserPulseInpComponent(int pulseLength, Pin.State normalState, Pin.State pulseState)

{

PulseLength = pulseLength;

NormalState = normalState;

PulseState = pulseState;

}

private int ticksLeft = 0;

public override void Tick()

{

if (ticksLeft > 0)

{

ticksLeft--;

GetOutpPin().SetState(PulseState);

} else

{

GetOutpPin().SetState(NormalState);

}

}

public static IComponent Constructor(string details)

{

int pulseLength, normalState, pulseState;

string[] detailArr = details.Split(',');

if (int.TryParse(detailArr[0], out pulseLength) && int.TryParse(detailArr[1], out normalState) && int.TryParse(detailArr[2], out pulseState))

{

return new UserPulseInpComponent(pulseLength, (Pin.State)normalState, (Pin.State)pulseState);

}

throw new PlacementException("Did not successfully parse int.");

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(0, 0.5F);

rect.X -= 0.5F;

rect.Width += 0.5F;

return rect;

}

public virtual ISettingDescription[] GetSettingDescriptions()

{

PulseLengthSettingDesc = new PositiveIntSettingDescription("How many ticks should this component pulse for?", PulseLength);

NormalStateSettingDesc = new EnumSettingDescription<Pin.State>("What state should this component output while not pulsing?", NormalState);

PulseStateSettingDesc = new EnumSettingDescription<Pin.State>("What state should this component output while pulsing?", PulseState);

return new ISettingDescription[] { PulseLengthSettingDesc, NormalStateSettingDesc, PulseStateSettingDesc };

}

public virtual void ApplySettings()

{

PulseLength = PulseLengthSettingDesc.GetValue();

NormalState = NormalStateSettingDesc.GetValue();

PulseState = PulseStateSettingDesc.GetValue();

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[] { new PointF(-0.5F, -0.5F), new PointF(0.5F, -0.5F), new PointF(1, 0), new PointF(0.5F, 0.5F), new PointF(-0.5F, 0.5F) });

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

public void Interact()

{

ticksLeft = PulseLength;

}

public override void ResetToDefault()

{

ticksLeft = 0;

}

}

### LogicProbeComponent

class LogicProbeComponent : SingInpNoneOutpBaseComponent, IGraphicalComponent

{

public override Pos GetInpOffset()

{

return new Pos(-2, 0);

}

public LogicProbeComponent()

{

DisplayFormat = new StringFormat

{

Alignment = StringAlignment.Center,

LineAlignment = StringAlignment.Center

};

}

public override void Tick() { }

public static string ID = "PROBE";

public static string DefaultDetails = "";

private Point? GraphicalLocation = null;

private float GraphicalScale = 1;

private StringFormat DisplayFormat;

public bool HasGraphics()

{

return true;

}

public Point? GetGraphicalElementLocation()

{

return GraphicalLocation;

}

public void SetGraphicalElementLocation(Point? location)

{

GraphicalLocation = location;

}

public override string GetComponentID()

{

return ID;

}

/\*

public override string GetComponentDetails()

{

using (Stream stream = new MemoryStream())

{

using (BinaryWriter bw = new BinaryWriter(stream))

{

bw.Write(GraphicalScale);

bw.Write(GraphicalLocation.HasValue);

if (GraphicalLocation.HasValue)

{

bw.Write(GraphicalLocation.Value.X);

bw.Write(GraphicalLocation.Value.Y);

}

stream.Position = 0;

using (BinaryReader br = new BinaryReader(stream))

{

return ByteEncoding.Byte.GetString(br.ReadBytes((int)stream.Length));

}

}

}

}

public static LogicProbeComponent Constructor(string details)

{

using (Stream stream = new MemoryStream(ByteEncoding.Byte.GetBytes(details)))

{

using (BinaryReader br = new BinaryReader(stream))

{

LogicProbeComponent retVal = new LogicProbeComponent();

retVal.GraphicalScale = br.ReadSingle();

if (br.ReadBoolean())

{

retVal.SetGraphicalElementLocation(new Point(br.ReadInt32(), br.ReadInt32()));

}

return retVal;

}

}

}

//\*/

public override string GetComponentDetails()

{

return "";

}

public static LogicProbeComponent Constructor(string details)

{

return new LogicProbeComponent();

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(1, 0.5F);

rect.Width -= 1.5F;

rect.X++;

return rect;

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[] { new PointF(0.5F, -0.5F), new PointF(-0.5F, -0.5F), new PointF(-1, 0), new PointF(-0.5F, 0.5F), new PointF(0.5F, 0.5F) });

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

public void RenderGraphicalElement(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

char display;

Color colour;

if (simulating)

{

Pin.State state = GetInpPin().GetStateForDisplay();

display = state.ToString()[0];

colour = colourScheme.GetWireColour(state);

} else

{

display = '?';

colour = colourScheme.Wire;

}

graphics.DrawString($"{display}", new Font("arial", 0.5F), new SolidBrush(colour), 0, 0, DisplayFormat);

}

public RectangleF GetGraphicalElementBounds()

{

return new RectangleF(-0.5F, -0.5F, 1, 1);

}

public float GetGraphicalElementScale()

{

return GraphicalScale;

}

public void SetGraphicalElementScale(float scale)

{

GraphicalScale = scale;

}

}

### SevenSegmentComponent

class SevenSegmentComponent : MultInpNoneOutpBaseComponent, IGraphicalComponent

{

public override RectangleF GetComponentBounds()

{

return new Rectangle(-4, -4, 7, 8);

}

public static string ID = "7SEG";

public static string DefaultDetails = "";

public override string GetComponentID()

{

return ID;

}

public bool HasGraphics()

{

return true;

}

/\*

public override string GetComponentDetails()

{

using (Stream stream = new MemoryStream())

{

using (BinaryWriter bw = new BinaryWriter(stream))

{

bw.Write(GraphicalScale);

bw.Write(GraphicalLocation.HasValue);

if (GraphicalLocation.HasValue)

{

bw.Write(GraphicalLocation.Value.X);

bw.Write(GraphicalLocation.Value.Y);

}

stream.Position = 0;

using (BinaryReader br = new BinaryReader(stream))

{

return ByteEncoding.Byte.GetString(br.ReadBytes((int)stream.Length));

}

}

}

}

public static SevenSegmentComponent Constructor(string details)

{

using (Stream stream = new MemoryStream(ByteEncoding.Byte.GetBytes(details)))

{

using (BinaryReader br = new BinaryReader(stream))

{

SevenSegmentComponent retVal = new SevenSegmentComponent();

retVal.GraphicalScale = br.ReadSingle();

if (br.ReadBoolean())

{

retVal.SetGraphicalElementLocation(new Point(br.ReadInt32(), br.ReadInt32()));

}

return retVal;

}

}

}

//\*/

public override string GetComponentDetails()

{

return "";

}

public static SevenSegmentComponent Constructor(string details)

{

return new SevenSegmentComponent();

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public RectangleF GetGraphicalElementBounds()

{

return new RectangleF(-2.25F, -3.75F, 4.5F, 7.5F);

}

public float GetGraphicalElementScale()

{

return GraphicalScale;

}

public void SetGraphicalElementScale(float scale)

{

GraphicalScale = scale;

}

private Point? GraphicalLocation = null;

private float GraphicalScale = 1;

public Point? GetGraphicalElementLocation()

{

return GraphicalLocation;

}

public override Pos[] GetInpOffsets()

{

return new Pos[]

{

new Pos(-4, -3),

new Pos(-4, -2),

new Pos(-4, -1),

new Pos(-4, 0),

new Pos(-4, 1),

new Pos(-4, 2),

new Pos(-4, 3)

};

}

private PointF[] GetDiamond(PointF around)

{

float dist = 0.75F;

return new PointF[]

{

new PointF(around.X - dist, around.Y),

new PointF(around.X, around.Y - dist),

new PointF(around.X + dist, around.Y),

new PointF(around.X, around.Y + dist)

};

}

private void DrawSegment(Graphics graphics, bool simulating, ColourScheme colourScheme, PointF point1, PointF point2, Pin.State state)

{

GraphicsPath path;

Brush brush = new SolidBrush(colourScheme.GetWireColour(state));

PointF[] dia1 = GetDiamond(point1), dia2 = GetDiamond(point2);

path = new GraphicsPath();

path.AddLines(dia1);

path.CloseFigure();

graphics.FillPath(brush, path);

path = new GraphicsPath();

path.AddLines(dia2);

path.CloseFigure();

graphics.FillPath(brush, path);

path = new GraphicsPath();

path.AddLines(new PointF[]

{

dia1[0],

dia1[2],

dia2[2],

dia2[0]

});

path.CloseFigure();

graphics.FillPath(brush, path);

path = new GraphicsPath();

path.AddLines(new PointF[]

{

dia1[1],

dia1[3],

dia2[3],

dia2[1]

});

path.CloseFigure();

graphics.FillPath(brush, path);

}

public void RenderGraphicalElement(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

Pin.State[] states;

if (simulating)

{

states = GetInpPins().Select(pin => pin.GetStateForDisplay()).ToArray();

} else

{

states = new Pin.State[7];

}

DrawSegment(graphics, simulating, colourScheme, new PointF(-0.5F, -3), new PointF(0.5F, -3), states[0]); // a

DrawSegment(graphics, simulating, colourScheme, new PointF(1.5F, -1), new PointF(1.5F, -2), states[1]); // b

DrawSegment(graphics, simulating, colourScheme, new PointF(1.5F, 1), new PointF(1.5F, 2), states[2]); // c

DrawSegment(graphics, simulating, colourScheme, new PointF(-0.5F, 3), new PointF(0.5F, 3), states[3]); // d

DrawSegment(graphics, simulating, colourScheme, new PointF(-1.5F, 1), new PointF(-1.5F, 2), states[4]); // e

DrawSegment(graphics, simulating, colourScheme, new PointF(-1.5F, -1), new PointF(-1.5F, -2), states[5]); // f

DrawSegment(graphics, simulating, colourScheme, new PointF(-0.5F, 0), new PointF(0.5F, 0), states[6]); // g

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

RectangleF bounds = GetComponentBounds();

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[]

{

new PointF(bounds.Left + 1, bounds.Top),

new PointF(bounds.Left + 1, bounds.Bottom),

new PointF(bounds.Right, bounds.Bottom),

new PointF(bounds.Right, bounds.Top)

});

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

public void SetGraphicalElementLocation(Point? location)

{

GraphicalLocation = location;

}

public override void Tick() { }

}

### BoardInputComponent

class BoardInputComponent : UserToggleInpComponent, IBoardInputComponent

{

private string ComponentName;

private Pin externalPin = null;

private NameSettingDescription nameSettingDesc;

private Board.InterfaceLocation interfaceLocation;

protected override string getOutputDescriptor()

{

return "default output";

}

public Board.InterfaceLocation GetInterfaceLocation()

{

return interfaceLocation;

}

public void SetInterfaceLocation(Board.InterfaceLocation location)

{

interfaceLocation = location;

}

public override Pos GetOutpOffset()

{

return new Pos(2, 0);

}

public string GetComponentName()

{

return ComponentName;

}

public void SetComponentName(string compName)

{

ComponentName = compName;

}

public BoardInputComponent(string name, Pin.State startDefaultState, Pin.State otherDefaultState) : base(startDefaultState, otherDefaultState)

{

ComponentName = name;

interfaceLocation = new Board.InterfaceLocation(Board.InterfaceLocation.SideEnum.Left, 0);

}

public BoardInputComponent(string name, Pin.State startDefaultState, Pin.State otherDefaultState, Board.InterfaceLocation interfaceLocation) : base(startDefaultState, otherDefaultState)

{

ComponentName = name;

this.interfaceLocation = interfaceLocation;

}

public new static string ID = "INPUT";

//public new static string DefaultDetails = $"I0,{(int)Pin.State.LOW},{(int)Pin.State.HIGH},{(byte)Board.InterfaceLocation.SideEnum.Left},{0}";

public new static string DefaultDetails = $"I0,{(int)Pin.State.LOW},{(int)Pin.State.HIGH}";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

//return $"{ComponentName},{(int)StartState},{(int)OtherState},{(byte)interfaceLocation.Side},{interfaceLocation.Distance}";

return $"{ComponentName},{(int)StartState},{(int)OtherState}";

}

public void SetExternalPin(Pin pin)

{

externalPin = pin;

OutputState = StartState;

}

public void RemoveExternalPin()

{

externalPin = null;

OutputState = StartState;

}

private bool tickAgain;

public override void Tick()

{

Pin.State prevState = GetOutpPin().GetStateForWireComponent();

if (externalPin != null)

{

OutputState = externalPin.GetStateForWireComponent();

}

base.Tick();

tickAgain = prevState != GetOutpPin().GetStateForWireComponent();

}

public bool TickAgain()

{

return tickAgain;

}

public new static BoardInputComponent Constructor(string details)

{

string[] strings = details.Split(',');

if (int.TryParse(strings[1], out int startDefaultState) && int.TryParse(strings[2], out int otherDefaultState) /\* && int.TryParse(strings[3], out int sideInt) && int.TryParse(strings[4], out int distInt) \*/)

{

return new BoardInputComponent(strings[0], (Pin.State)startDefaultState, (Pin.State)otherDefaultState /\*, new Board.InterfaceLocation((Board.InterfaceLocation.SideEnum)(byte)sideInt, distInt) \*/);

}

throw new PlacementException("Did not successfully parse int.");

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override ISettingDescription[] GetSettingDescriptions()

{

nameSettingDesc = new NameSettingDescription("What is this component called?", ComponentName);

return (new ISettingDescription[] { nameSettingDesc }).Concat(base.GetSettingDescriptions()).ToArray();

}

public override void ApplySettings()

{

base.ApplySettings();

ComponentName = nameSettingDesc.GetValue();

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = base.GetComponentBounds();

rect.Width++;

rect.X--;

return rect;

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

base.Render(graphics, simulating, colourScheme);

graphics.DrawString(ComponentName, new Font("arial", 0.5F), Brushes.Black, -1, -0.25F);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[] { new PointF(-1.5F, -0.5F), new PointF(0.5F, -0.5F), new PointF(1, 0), new PointF(0.5F, 0.5F), new PointF(-1.5F, 0.5F) });

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

}

### BoardOutputComponent

class BoardOutputComponent : SingInpNoneOutpBaseComponent, IBoardOutputComponent, ISettingsComponent

{

private string ComponentName;

//protected Pin.State State;

private Pin externalPin = null;

private NameSettingDescription nameSettingDesc;

private Board.InterfaceLocation interfaceLocation;

public Board.InterfaceLocation GetInterfaceLocation()

{

return interfaceLocation;

}

public void SetInterfaceLocation(Board.InterfaceLocation location)

{

interfaceLocation = location;

}

public override Pos GetInpOffset()

{

return new Pos(-2, 0);

}

public string GetComponentName()

{

return ComponentName;

}

public void SetComponentName(string compName)

{

ComponentName = compName;

}

public BoardOutputComponent(string name)

{

ComponentName = name;

interfaceLocation = new Board.InterfaceLocation(Board.InterfaceLocation.SideEnum.Right, 0);

}

public BoardOutputComponent(string name, Board.InterfaceLocation interfaceLocation)

{

ComponentName = name;

this.interfaceLocation = interfaceLocation;

}

public static string ID = "OUTPUT";

//public static string DefaultDetails = $"O0,{(byte)Board.InterfaceLocation.SideEnum.Right},{0}";

public static string DefaultDetails = "O0";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

//return $"{ComponentName},{(byte)interfaceLocation.Side},{interfaceLocation.Distance}";

return ComponentName;

}

public static BoardOutputComponent Constructor(string details)

{

/\*

string[] strings = details.Split(',');

if (int.TryParse(strings[1], out int sideInt) && int.TryParse(strings[2], out int distInt))

{

return new BoardOutputComponent(strings[0], new Board.InterfaceLocation((Board.InterfaceLocation.SideEnum)(byte)sideInt, distInt));

}

throw new PlacementException("Did not successfully parse int.");

//\*/ return new BoardOutputComponent(details);

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public void SetExternalPin(Pin pin)

{

externalPin = pin;

}

public void RemoveExternalPin()

{

externalPin = null;

}

private bool tickAgain;

public override void Tick()

{

tickAgain = false;

if (externalPin != null)

{

Pin.State prevState = externalPin.GetStateForWireComponent();

externalPin.SetState(GetInpPin().GetStateForWireComponent());

tickAgain = prevState != externalPin.GetStateForWireComponent();

}

}

public bool TickAgain()

{

return tickAgain;

}

public ISettingDescription[] GetSettingDescriptions()

{

nameSettingDesc = new NameSettingDescription("What is this component called?", ComponentName);

return new ISettingDescription[] { nameSettingDesc };

}

public void ApplySettings()

{

ComponentName = nameSettingDesc.GetValue();

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(1, 0.5F);

rect.Width -= 0.5F;

rect.X++;

return rect;

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

base.Render(graphics, simulating, colourScheme);

graphics.DrawString(ComponentName, new Font("arial", 0.5F), Brushes.Black, -0.5F, -0.25F);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[] { new PointF(1.5F, -0.5F), new PointF(-0.5F, -0.5F), new PointF(-1, 0), new PointF(-0.5F, 0.5F), new PointF(1.5F, 0.5F) });

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

}

### BoardBidirComponent

class BoardBidirComponent : SingInpSingOutpBaseComponent, IBoardInputComponent, IBoardOutputComponent, ISettingsComponent

{

private string ComponentName;

private Pin.State DefaultExternalState;

private Pin externalPin = null;

private NameSettingDescription nameSettingDesc;

private EnumSettingDescription<Pin.State> defaultStateSettingDesc;

private Board.InterfaceLocation interfaceLocation;

public Board.InterfaceLocation GetInterfaceLocation()

{

return interfaceLocation;

}

public void SetInterfaceLocation(Board.InterfaceLocation location)

{

interfaceLocation = location;

}

private readonly Pos offset = new Pos(0, 2);

public override Pos GetInpOffset()

{

return offset;

}

public override Pos GetOutpOffset()

{

return offset;

}

public string GetComponentName()

{

return ComponentName;

}

public void SetComponentName(string compName)

{

ComponentName = compName;

}

public BoardBidirComponent(string name, Pin.State defaultState)

{

ComponentName = name;

DefaultExternalState = defaultState;

interfaceLocation = new Board.InterfaceLocation(Board.InterfaceLocation.SideEnum.Top, 0);

}

public BoardBidirComponent(string name, Pin.State defaultState, Board.InterfaceLocation interfaceLocation)

{

ComponentName = name;

DefaultExternalState = defaultState;

this.interfaceLocation = interfaceLocation;

}

public static string ID = "BIDIR";

//public static string DefaultDetails = $"B0,{(int)Pin.State.LOW},{(byte)Board.InterfaceLocation.SideEnum.Top},{0}";

public static string DefaultDetails = "B0";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

//return $"{ComponentName},{(int)DefaultExternalState},{(byte)interfaceLocation.Side},{interfaceLocation.Distance}";

return ComponentName;

}

public void SetExternalPin(Pin pin)

{

externalPin = pin;

}

public void RemoveExternalPin()

{

externalPin = null;

}

private bool tickAgain;

public override void Tick()

{

Pin.State prevIntState = GetOutpPin().GetStateForWireComponent();

if (externalPin is null)

{

GetOutpPin().SetState(DefaultExternalState);

tickAgain = false;

} else

{

Pin.State prevExtState = externalPin.GetStateForWireComponent();

externalPin.SetState(prevIntState);

GetOutpPin().SetState(prevExtState);

tickAgain = prevExtState != externalPin.GetStateForWireComponent();

}

tickAgain |= prevIntState != GetOutpPin().GetStateForWireComponent();

}

public bool TickAgain()

{

return tickAgain;

}

public static BoardBidirComponent Constructor(string details)

{

string[] strings = details.Split(',');

if (int.TryParse(strings[1], out int stateInt) /\* && int.TryParse(strings[2], out int sideInt) && int.TryParse(strings[3], out int distInt) \*/)

{

return new BoardBidirComponent(strings[0], (Pin.State)stateInt /\*, new Board.InterfaceLocation((Board.InterfaceLocation.SideEnum)sideInt, distInt) \*/);

}

throw new PlacementException("Did not successfully parse int.");

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public ISettingDescription[] GetSettingDescriptions()

{

nameSettingDesc = new NameSettingDescription("What is this component called?", ComponentName);

defaultStateSettingDesc = new EnumSettingDescription<Pin.State>("What is the simulated external state of this component?", DefaultExternalState);

return new ISettingDescription[] { nameSettingDesc, defaultStateSettingDesc };

}

public void ApplySettings()

{

ComponentName = nameSettingDesc.GetValue();

DefaultExternalState = defaultStateSettingDesc.GetValue();

}

public override RectangleF GetComponentBounds()

{

RectangleF rect = GetDefaultComponentBounds();

rect.Inflate(0.5F, 1);

rect.Height -= 0.5F;

rect.Y -= 0.5F;

return rect;

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

InpOutpTools.DrawInpOutpLine(graphics, simulating, offset, new PointF(offset.X, offset.Y - 1.5F), colourScheme, this);

RenderMainShape(graphics, simulating, colourScheme);

graphics.DrawString(ComponentName, new Font("arial", 0.5F), Brushes.Black, -1, -0.25F);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddLines(new PointF[] { new PointF(-0.5F, -1.5F), new PointF(-0.5F, 0.5F), new PointF(0, 1), new PointF(0.5F, 0.5F), new PointF(0.5F, -1.5F) });

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

}

}

### BoardContainerComponent

class BoardContainerComponent : MultInpMultOutpBaseComponent, IBoardContainerComponent

{

private bool isInitialized = false;

private string InternalBoardName;

public Board InternalBoard { get; private set; }

private Rectangle Shape;

private Pos[] InpOffsets;

private string[] InpNames;

private Pos[] OutpOffsets;

private string[] OutpNames;

private static StringFormat LeftStringFormat, RightStringFormat, TopStringFormat, BottomStringFormat;

private static Dictionary<Board.InterfaceLocation.SideEnum, StringFormat> StringFormats;

private Point? GraphicalLocation = null;

private float GraphicalScale = 1;

public override Pos[] GetInpOffsets()

{

return InpOffsets.ToArray();

}

public override Pos[] GetOutpOffsets()

{

return OutpOffsets.ToArray();

}

static BoardContainerComponent()

{

LeftStringFormat = new StringFormat();

RightStringFormat = new StringFormat();

TopStringFormat = new StringFormat();

BottomStringFormat = new StringFormat();

LeftStringFormat.Alignment = StringAlignment.Near;

TopStringFormat.Alignment = StringAlignment.Center;

BottomStringFormat.Alignment = StringAlignment.Center;

RightStringFormat.Alignment = StringAlignment.Far;

TopStringFormat.LineAlignment = StringAlignment.Near;

LeftStringFormat.LineAlignment = StringAlignment.Center;

RightStringFormat.LineAlignment = StringAlignment.Center;

BottomStringFormat.LineAlignment = StringAlignment.Far;

StringFormats = new Dictionary<Board.InterfaceLocation.SideEnum, StringFormat>

{

{ Board.InterfaceLocation.SideEnum.Left, LeftStringFormat },

{ Board.InterfaceLocation.SideEnum.Right, RightStringFormat },

{ Board.InterfaceLocation.SideEnum.Top, TopStringFormat },

{ Board.InterfaceLocation.SideEnum.Bottom, BottomStringFormat }

};

}

public void ProvideInternalBoard(Board board)

{

if (InternalBoard == null)

{

InternalBoard = board;

InternalBoardName = board.Name;

Initialize();

if (IsPlaced())

{

PlaceForInternalBoard();

}

}

}

public BoardContainerComponent(string boardName)

{

InternalBoardName = boardName;

if (IsPlaced())

{

try

{

ProvideInternalBoard(GetComponentBoard().GetTopLevelBoard().GetBoardList().First(board => board.Name == boardName));

return;

} catch (InvalidOperationException) { }

}

//ReadWriteImplementation.PromiseBoard(boardName, ProvideInternalBoard);

}

public BoardContainerComponent(Board internalBoard, bool copy = true)

{

InternalBoard = copy ? internalBoard.Copy() : internalBoard;

InternalBoardName = internalBoard.Name;

Initialize();

}

private List<Action<IBoardContainerComponent>> detailsProviders = new List<Action<IBoardContainerComponent>>();

public void PromiseDetails(Action<IBoardContainerComponent> detailsProvider)

{

if (isInitialized)

{

detailsProvider(this);

} else

{

detailsProviders.Add(detailsProvider);

}

}

private void Initialize()

{

isInitialized = true;

InternalBoard.SizeChanged += InternalBoard\_SizeChanged;

ResetShape();

IBoardInterfaceComponent[] interfaceComps = InternalBoard.GetInterfaceComponents();

List<Pos> inpOffsetList = new List<Pos>(), outpOffsetList = new List<Pos>();

List<string> inpNameList = new List<string>(), outpNameList = new List<string>();

Board.InterfaceLocation interfaceLocation;

Pos offset;

(Pos, Pos) offsetInfo;

foreach (IBoardInterfaceComponent interfaceComp in interfaceComps)

{

interfaceLocation = interfaceComp.GetInterfaceLocation();

offsetInfo = GetOffset(interfaceLocation);

offset = offsetInfo.Item1.Add(offsetInfo.Item2);

if (interfaceComp is IBoardInputComponent)

{

inpOffsetList.Add(offset);

inpNameList.Add(interfaceComp.GetComponentName());

}

if (interfaceComp is IBoardOutputComponent)

{

outpOffsetList.Add(offset);

outpNameList.Add(interfaceComp.GetComponentName());

}

}

InpOffsets = inpOffsetList.ToArray();

InpNames = inpNameList.ToArray();

OutpOffsets = outpOffsetList.ToArray();

OutpNames = outpNameList.ToArray();

while (detailsProviders.Count > 0)

{

detailsProviders[0](this);

detailsProviders.RemoveAt(0);

}

}

private void InternalBoard\_SizeChanged()

{

ResetShape();

}

public void ResetShape()

{

Shape = new Rectangle(-InternalBoard.ExternalSize.Width / 2, -InternalBoard.ExternalSize.Height / 2, InternalBoard.ExternalSize.Width, InternalBoard.ExternalSize.Height);

}

public static string ID = "BOARD";

public override string GetComponentID()

{

return ID;

}

public override string GetComponentDetails()

{

return GetInternalBoardName();

}

private (Pos, Pos) GetOffset(Board.InterfaceLocation interfaceLocation)

{

if (interfaceLocation.Side.IsTop())

{

return (new Pos(Shape.Left + interfaceLocation.Distance, Shape.Top), new Pos(0, -1));

}

else if (interfaceLocation.Side.IsBottom())

{

return (new Pos(Shape.Left + interfaceLocation.Distance, Shape.Bottom), new Pos(0, 1));

}

else if (interfaceLocation.Side.IsLeft())

{

return (new Pos(Shape.Left, Shape.Top + interfaceLocation.Distance), new Pos(-1, 0));

}

else if (interfaceLocation.Side.IsRight())

{

return (new Pos(Shape.Right, Shape.Top + interfaceLocation.Distance), new Pos(1, 0));

}

return (new Pos(), new Pos());

}

public override void Place(Pos pos, Rotation rotation, Board board)

{

base.Place(pos, rotation, board);

if (InternalBoard != null)

{

PlaceForInternalBoard();

}

}

private void PlaceForInternalBoard()

{

Pin[] inpPins = GetInpPins();

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

{

InternalBoard.GetInputComponent(InpNames[i]).SetExternalPin(inpPins[i]);

}

Pin[] outpPins = GetOutpPins();

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

{

InternalBoard.GetOutputComponent(OutpNames[i]).SetExternalPin(outpPins[i]);

}

InternalBoard.SetOwnerBoard(GetComponentBoard());

}

public override void Remove()

{

base.Remove();

if (isInitialized)

{

foreach (IBoardInputComponent inpComp in InternalBoard.GetInputComponents())

{

inpComp.RemoveExternalPin();

}

foreach (IBoardOutputComponent outpComp in InternalBoard.GetOutputComponents())

{

outpComp.RemoveExternalPin();

}

}

InternalBoard.ResetOwnerBoard();

}

public override void Tick()

{

InternalBoard.TickComponents();

}

public static BoardContainerComponent Constructor(string details)

{

return new BoardContainerComponent(details);

}

public override IComponent NonStaticConstructor(string details)

{

return Constructor(details);

}

public override RectangleF GetComponentBounds()

{

Pos[] offsets = new Pos[] { new Pos(0, 0), new Pos(Shape.Left, Shape.Top), new Pos(Shape.Right, Shape.Bottom) };

if (InternalBoard != null)

{

offsets = offsets.Concat(GetAllPinOffsets()).ToArray();

}

RectangleF rect = CreateSmallestRect(offsets);

return rect;

}

public override void Render(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

Pos otherOffset;

(Pos, Pos) offsetInfo;

foreach (IBoardInterfaceComponent interfaceComp in InternalBoard.GetInterfaceComponents())

{

offsetInfo = GetOffset(interfaceComp.GetInterfaceLocation());

otherOffset = offsetInfo.Item1.Add(new Pos(-offsetInfo.Item2.X, -offsetInfo.Item2.Y));

InpOutpTools.DrawInpOutpLine(graphics, simulating, offsetInfo.Item1.Add(offsetInfo.Item2), new PointF(otherOffset.X, otherOffset.Y), colourScheme, this);

}

RenderMainShape(graphics, simulating, colourScheme);

}

public override void RenderMainShape(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

GraphicsPath path = new GraphicsPath();

path.AddRectangle(Shape);

path.CloseFigure();

DrawComponentFromPath(graphics, path, colourScheme);

(Pos, Pos) offsetInfo;

foreach (IBoardInterfaceComponent interfaceComp in InternalBoard.GetInterfaceComponents())

{

Board.InterfaceLocation interfaceLocation = interfaceComp.GetInterfaceLocation();

offsetInfo = GetOffset(interfaceLocation);

graphics.DrawString(interfaceComp.GetComponentName(), new Font("arial", 0.25F), Brushes.Black, new Point(offsetInfo.Item1.X, offsetInfo.Item1.Y), StringFormats[interfaceLocation.Side]);

}

}

public void RenderGraphicalElement(Graphics graphics, bool simulating, ColourScheme colourScheme)

{

Matrix matrix;

PointF? loc;

float scale;

foreach (IGraphicalComponent graphicalComp in InternalBoard.GetGraphicalComponents().Where(comp => comp.HasGraphics()))

{

loc = graphicalComp.GetGraphicalElementLocation();

scale = graphicalComp.GetGraphicalElementScale();

if (loc.HasValue)

{

using (new TransformRestorer(graphics))

{

matrix = new Matrix();

matrix.Translate(loc.Value.X, loc.Value.Y);

matrix.Scale(scale, scale);

graphics.MultiplyTransform(matrix);

graphicalComp.RenderGraphicalElement(graphics, simulating, colourScheme);

}

}

}

}

public Rectangle GetShape()

{

return Shape;

}

public string GetInternalBoardName()

{

return InternalBoard?.Name ?? InternalBoardName;

}

public Board GetInternalBoard()

{

return InternalBoard;

}

public bool HasGraphics()

{

return InternalBoard?.GetGraphicalComponents().Any(comp => comp.HasGraphics()) ?? false;

}

public Point? GetGraphicalElementLocation()

{

return GraphicalLocation;

}

public void SetGraphicalElementLocation(Point? point)

{

GraphicalLocation = point;

}

public RectangleF GetGraphicalElementBounds()

{

RectangleF rect = new RectangleF();

RectangleF? compRect;

foreach (IGraphicalComponent graphicalComp in InternalBoard.GetGraphicalComponents())

{

compRect = graphicalComp.GetOffsetGraphicalElementBounds();

if (compRect.HasValue)

{

rect = RectangleF.FromLTRB(

Math.Min(rect.Left, compRect.Value.Left),

Math.Min(rect.Top, compRect.Value.Top),

Math.Max(rect.Right, compRect.Value.Right),

Math.Max(rect.Bottom, compRect.Value.Bottom)

);

}

}

return rect;

}

public float GetGraphicalElementScale()

{

return GraphicalScale;

}

public void SetGraphicalElementScale(float scale)

{

GraphicalScale = scale;

}

public override void ResetToDefault()

{

InternalBoard.ResetForSimulation();

}

public override string ToString()

{

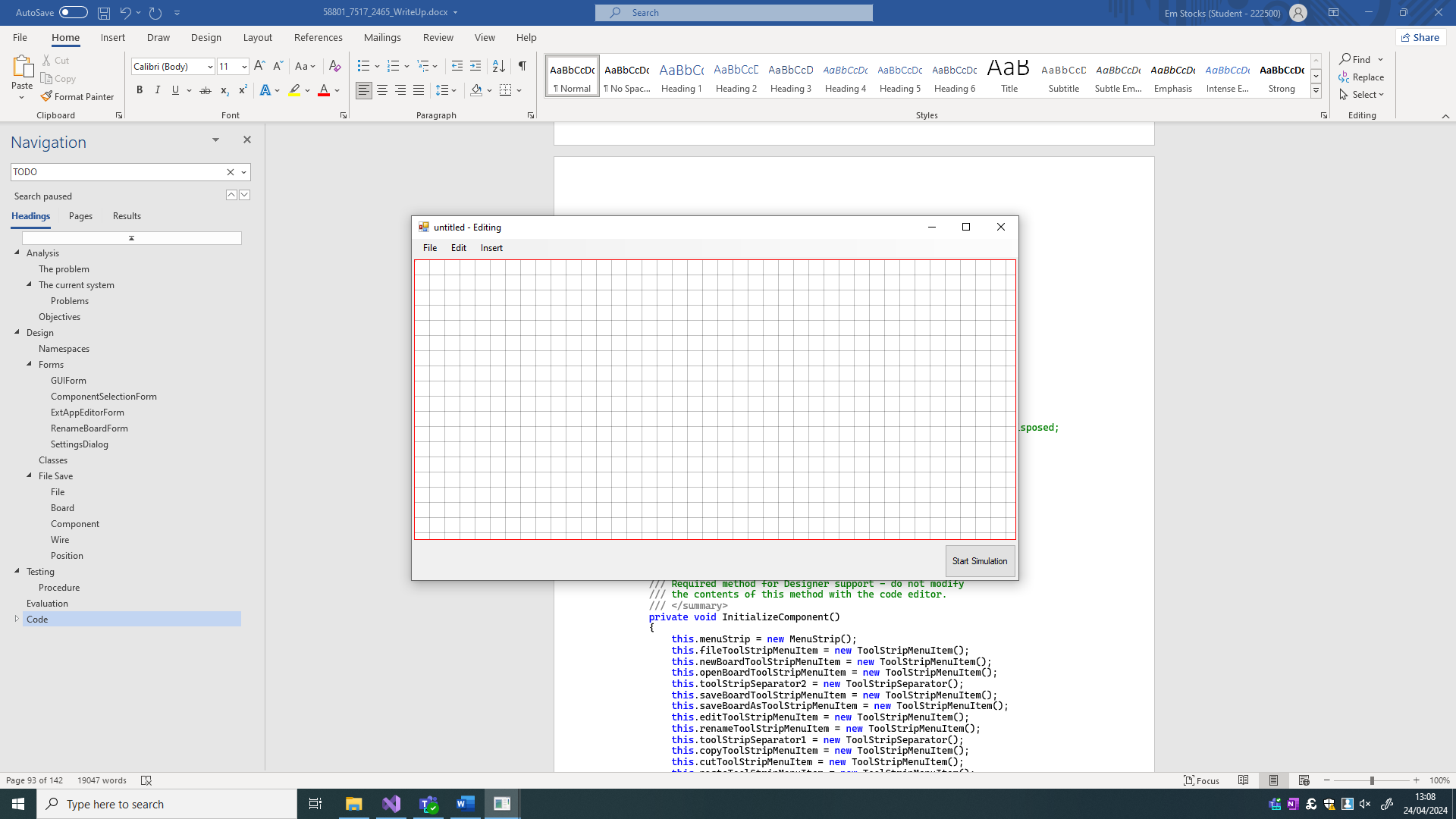
return $"{GetComponentID()}:{InternalBoardName}@{GetComponentPos()}";

}

}

## GUI

### GUIForm



partial class GUIForm

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.menuStrip = new MenuStrip();

this.fileToolStripMenuItem = new ToolStripMenuItem();

this.newBoardToolStripMenuItem = new ToolStripMenuItem();

this.openBoardToolStripMenuItem = new ToolStripMenuItem();

this.toolStripSeparator2 = new ToolStripSeparator();

this.saveBoardToolStripMenuItem = new ToolStripMenuItem();

this.saveBoardAsToolStripMenuItem = new ToolStripMenuItem();

this.editToolStripMenuItem = new ToolStripMenuItem();

this.renameToolStripMenuItem = new ToolStripMenuItem();

this.toolStripSeparator1 = new ToolStripSeparator();

this.copyToolStripMenuItem = new ToolStripMenuItem();

this.cutToolStripMenuItem = new ToolStripMenuItem();

this.pasteToolStripMenuItem = new ToolStripMenuItem();

this.toolStripSeparator3 = new ToolStripSeparator();

this.editExternalAppearanceToolStripMenuItem = new ToolStripMenuItem();

this.insertToolStripMenuItem = new ToolStripMenuItem();

this.insertBuiltinComponentToolStripMenuItem = new ToolStripMenuItem();

this.insertBoardComponentToolStripMenuItem = new ToolStripMenuItem();

this.menuStrip.SuspendLayout();

this.tblWholePage = new TableLayoutPanel();

this.builder = Builder.NewBoard("untitled");

this.btnSimulate = new Button();

this.tblSelector = new TableLayoutPanel();

this.tblWholePage.SuspendLayout();

this.SuspendLayout();

//

// menuStrip

//

this.menuStrip.Items.AddRange(new ToolStripItem[] {

this.fileToolStripMenuItem,

this.editToolStripMenuItem,

this.insertToolStripMenuItem});

this.menuStrip.Location = new Point(0, 0);

this.menuStrip.Name = "menuStrip";

this.menuStrip.Size = new Size(800, 24);

this.menuStrip.TabIndex = 0;

this.menuStrip.Text = "menuStrip";

//

// fileToolStripMenuItem

//

this.fileToolStripMenuItem.DropDownItems.AddRange(new ToolStripItem[] {

this.newBoardToolStripMenuItem,

this.openBoardToolStripMenuItem,

this.toolStripSeparator2,

this.saveBoardToolStripMenuItem,

this.saveBoardAsToolStripMenuItem});

this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";

this.fileToolStripMenuItem.Size = new Size(37, 20);

this.fileToolStripMenuItem.Text = "File";

//

// newBoardToolStripMenuItem

//

this.newBoardToolStripMenuItem.Name = "newBoardToolStripMenuItem";

this.newBoardToolStripMenuItem.Size = new Size(180, 22);

this.newBoardToolStripMenuItem.Text = "New Board";

this.newBoardToolStripMenuItem.Click += new EventHandler(this.newBoardToolStripMenuItem\_Click);

//

// openBoardToolStripMenuItem

//

this.openBoardToolStripMenuItem.Name = "openBoardToolStripMenuItem";

this.openBoardToolStripMenuItem.Size = new Size(180, 22);

this.openBoardToolStripMenuItem.Text = "Open Board";

this.openBoardToolStripMenuItem.Click += new EventHandler(this.openBoardToolStripMenuItem\_Click);

//

// toolStripSeparator2

//

this.toolStripSeparator2.Name = "toolStripSeparator2";

this.toolStripSeparator2.Size = new Size(177, 6);

//

// saveBoardToolStripMenuItem

//

this.saveBoardToolStripMenuItem.Name = "saveBoardToolStripMenuItem";

this.saveBoardToolStripMenuItem.Size = new Size(180, 22);

this.saveBoardToolStripMenuItem.Text = "Save Board";

this.saveBoardToolStripMenuItem.Click += new EventHandler(this.saveBoardToolStripMenuItem\_Click);

//

// saveBoardAsToolStripMenuItem

//

this.saveBoardAsToolStripMenuItem.Name = "saveBoardAsToolStripMenuItem";

this.saveBoardAsToolStripMenuItem.Size = new Size(180, 22);

this.saveBoardAsToolStripMenuItem.Text = "Save Board As";

this.saveBoardAsToolStripMenuItem.Click += new EventHandler(this.saveBoardAsToolStripMenuItem\_Click);

//

// editToolStripMenuItem

//

this.editToolStripMenuItem.DropDownItems.AddRange(new ToolStripItem[] {

this.renameToolStripMenuItem,

this.toolStripSeparator1,

this.copyToolStripMenuItem,

this.cutToolStripMenuItem,

this.pasteToolStripMenuItem,

this.toolStripSeparator3,

this.editExternalAppearanceToolStripMenuItem});

this.editToolStripMenuItem.Name = "editToolStripMenuItem";

this.editToolStripMenuItem.Size = new Size(39, 20);

this.editToolStripMenuItem.Text = "Edit";

//

// renameToolStripMenuItem

//

this.renameToolStripMenuItem.Name = "renameToolStripMenuItem";

this.renameToolStripMenuItem.Size = new Size(205, 22);

this.renameToolStripMenuItem.Text = "Rename";

this.renameToolStripMenuItem.Click += new EventHandler(this.renameToolStripMenuItem\_Click);

//

// toolStripSeparator1

//

this.toolStripSeparator1.Name = "toolStripSeparator1";

this.toolStripSeparator1.Size = new Size(202, 6);

//

// copyToolStripMenuItem

//

this.copyToolStripMenuItem.Name = "copyToolStripMenuItem";

this.copyToolStripMenuItem.Size = new Size(205, 22);

this.copyToolStripMenuItem.Text = "Copy";

this.copyToolStripMenuItem.Click += new EventHandler(this.copyToolStripMenuItem\_Click);

//

// cutToolStripMenuItem

//

this.cutToolStripMenuItem.Name = "cutToolStripMenuItem";

this.cutToolStripMenuItem.Size = new Size(205, 22);

this.cutToolStripMenuItem.Text = "Cut";

this.cutToolStripMenuItem.Click += new EventHandler(this.cutToolStripMenuItem\_Click);

//

// pasteToolStripMenuItem

//

this.pasteToolStripMenuItem.Name = "pasteToolStripMenuItem";

this.pasteToolStripMenuItem.Size = new Size(205, 22);

this.pasteToolStripMenuItem.Text = "Paste";

this.pasteToolStripMenuItem.Click += new EventHandler(this.pasteToolStripMenuItem\_Click);

//

// toolStripSeparator3

//

this.toolStripSeparator3.Name = "toolStripSeparator3";

this.toolStripSeparator3.Size = new Size(202, 6);

//

// editExternalAppearanceToolStripMenuItem

//

this.editExternalAppearanceToolStripMenuItem.Name = "editExternalAppearanceToolStripMenuItem";

this.editExternalAppearanceToolStripMenuItem.Size = new Size(205, 22);

this.editExternalAppearanceToolStripMenuItem.Text = "Edit External Appearance";

this.editExternalAppearanceToolStripMenuItem.Click += new EventHandler(this.editExternalAppearanceToolStripMenuItem\_Click);

//

// insertToolStripMenuItem

//

this.insertToolStripMenuItem.DropDownItems.AddRange(new ToolStripItem[] {

this.insertBuiltinComponentToolStripMenuItem,

this.insertBoardComponentToolStripMenuItem});

this.insertToolStripMenuItem.Name = "insertToolStripMenuItem";

this.insertToolStripMenuItem.Size = new Size(48, 20);

this.insertToolStripMenuItem.Text = "Insert";

//

// insertComponentToolStripMenuItem

//

this.insertBuiltinComponentToolStripMenuItem.Name = "insertComponentToolStripMenuItem";

this.insertBuiltinComponentToolStripMenuItem.Size = new Size(180, 22);

this.insertBuiltinComponentToolStripMenuItem.Text = "Insert Builtin Component";

this.insertBuiltinComponentToolStripMenuItem.Click += new EventHandler(this.insertBuiltinComponentToolStripMenuItem\_Click);

//

// insertFromBoardToolStripMenuItem

//

this.insertBoardComponentToolStripMenuItem.Name = "insertFromBoardToolStripMenuItem";

this.insertBoardComponentToolStripMenuItem.Size = new Size(180, 22);

this.insertBoardComponentToolStripMenuItem.Text = "Insert Board Component";

this.insertBoardComponentToolStripMenuItem.Click += new EventHandler(this.insertBoardComponentToolStripMenuItem\_Click);

//

// tblWholePage

//

this.tblWholePage.ColumnCount = 2;

this.tblWholePage.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 100F));

this.tblWholePage.ColumnStyles.Add(new ColumnStyle(SizeType.Absolute, 100F));

this.tblWholePage.Controls.Add(this.builder, 0, 0);

this.tblWholePage.Controls.Add(this.btnSimulate, 1, 1);

this.tblWholePage.Controls.Add(this.tblSelector, 0, 1);

this.tblWholePage.Dock = DockStyle.Fill;

this.tblWholePage.Location = new Point(0, 24);

this.tblWholePage.Name = "tblWholePage";

this.tblWholePage.RowCount = 2;

this.tblWholePage.RowStyles.Add(new RowStyle(SizeType.Percent, 100F));

this.tblWholePage.RowStyles.Add(new RowStyle(SizeType.Absolute, 50F));

this.tblWholePage.Size = new Size(800, 426);

this.tblWholePage.TabIndex = 2;

//

// builder

//

this.tblWholePage.SetColumnSpan(this.builder, 2);

this.builder.Dock = DockStyle.Fill;

this.builder.Location = new Point(3, 3);

this.builder.Name = "builder";

this.builder.Size = new Size(794, 370);

this.builder.TabIndex = 0;

this.builder.SimulatingChange += BtnSimulate\_UpdateText;

this.builder.SimulatingChange += Builder\_SimulatingChange;

//

// btnSimulate

//

this.btnSimulate.Dock = DockStyle.Fill;

this.btnSimulate.Location = new Point(703, 379);

this.btnSimulate.Name = "btnSimulate";

this.btnSimulate.Size = new Size(94, 44);

this.btnSimulate.TabIndex = 1;

this.btnSimulate.Text = "Start Simulation";

this.btnSimulate.UseVisualStyleBackColor = true;

this.btnSimulate.Click += BtnSimulate\_Click;

//

// tblSelector

//

this.tblSelector.ColumnCount = 2;

this.tblSelector.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F));

this.tblSelector.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F));

this.tblSelector.Dock = DockStyle.Bottom;

this.tblSelector.Location = new Point(3, 379);

this.tblSelector.Name = "tblSelector";

this.tblSelector.RowCount = 1;

this.tblSelector.RowStyles.Add(new RowStyle(SizeType.Percent, 50F));

this.tblSelector.Size = new Size(694, 44);

this.tblSelector.TabIndex = 2;

//

// GUIForm

//

this.AutoScaleDimensions = new SizeF(6F, 13F);

this.AutoScaleMode = AutoScaleMode.Font;

this.ClientSize = new Size(800, 450);

this.Controls.Add(this.tblWholePage);

this.Controls.Add(this.menuStrip);

this.Name = "GUIForm";

this.Text = "GUIForm";

this.menuStrip.ResumeLayout(false);

this.menuStrip.PerformLayout();

this.tblWholePage.ResumeLayout(false);

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private MenuStrip menuStrip;

private ToolStripMenuItem fileToolStripMenuItem;

private ToolStripMenuItem newBoardToolStripMenuItem;

private ToolStripMenuItem openBoardToolStripMenuItem;

private ToolStripMenuItem saveBoardToolStripMenuItem;

private ToolStripMenuItem saveBoardAsToolStripMenuItem;

private ToolStripMenuItem editToolStripMenuItem;

private ToolStripMenuItem renameToolStripMenuItem;

private ToolStripMenuItem copyToolStripMenuItem;

private ToolStripMenuItem cutToolStripMenuItem;

private ToolStripMenuItem pasteToolStripMenuItem;

private ToolStripSeparator toolStripSeparator2;

private ToolStripSeparator toolStripSeparator1;

private ToolStripSeparator toolStripSeparator3;

private ToolStripMenuItem editExternalAppearanceToolStripMenuItem;

private ToolStripMenuItem insertToolStripMenuItem;

private ToolStripMenuItem insertBuiltinComponentToolStripMenuItem;

private ToolStripMenuItem insertBoardComponentToolStripMenuItem;

private TableLayoutPanel tblWholePage;

private Builder builder;

private Button btnSimulate;

private TableLayoutPanel tblSelector;

}

public partial class GUIForm : Form

{

public GUIForm()

{

InitializeComponent();

UpdateFormText();

}

private void newBoardToolStripMenuItem\_Click(object sender, EventArgs e)

{

builder.OpenNewBoard("untitled");

UpdateFormText();

Invalidate();

}

private void openBoardToolStripMenuItem\_Click(object sender, EventArgs e)

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.Filter = "Boards (\*.brd)|\*.brd|All files (\*.\*)|\*.\*";

openFileDialog.FilterIndex = 0;

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

builder.OpenLoadBoard(openFileDialog.FileName);

UpdateFormText();

}

}

Invalidate();

}

private void saveBoardToolStripMenuItem\_Click(object sender, EventArgs e)

{

builder.SaveBoard();

}

private void saveBoardAsToolStripMenuItem\_Click(object sender, EventArgs e)

{

using (SaveFileDialog saveFileDialog = new SaveFileDialog())

{

saveFileDialog.Filter = "Boards (\*.brd)|\*.brd|All files (\*.\*)|\*.\*";

saveFileDialog.FilterIndex = 0;

saveFileDialog.FileName = builder.GetBoard().Name;

saveFileDialog.DefaultExt = "brd";

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

string name = saveFileDialog.FileName.Split('\\').LastOrDefault();

if (name.EndsWith(".brd"))

{

name = name.Remove(name.Length - 4);

}

builder.GetBoard().Name = name;

UpdateFormText();

builder.SaveBoard(saveFileDialog.FileName);

}

}

Invalidate();

}

private void renameToolStripMenuItem\_Click(object sender, EventArgs e)

{

RenameBoardForm dialog = new RenameBoardForm(builder.GetBoard().Name);

if (dialog.ShowDialog() == DialogResult.OK)

{

builder.GetBoard().Name = dialog.NewBoardName();

UpdateFormText();

}

}

private void cutToolStripMenuItem\_Click(object sender, EventArgs e)

{

builder.CutSelection();

Invalidate();

}

private void copyToolStripMenuItem\_Click(object sender, EventArgs e)

{

builder.CopySelection();

Invalidate();

}

private void pasteToolStripMenuItem\_Click(object sender, EventArgs e)

{

builder.PasteSelection();

Invalidate();

}

private void editExternalAppearanceToolStripMenuItem\_Click(object sender, EventArgs e)

{

ExtAppEditorForm dialog = new ExtAppEditorForm(new BoardContainerComponent(builder.GetBoard(), false), builder.colourScheme);

if (dialog.ShowDialog() != DialogResult.OK)

{

dialog.ResetChanges();

}

}

private void insertBuiltinComponentToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

builder.CreateBuiltinComponent();

}

catch (PlacementException ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void insertBoardComponentToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

builder.CreateBoardComponent();

} catch (PlacementException ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void BtnSimulate\_Click(object sender, EventArgs e)

{

builder.SetSimulation(!builder.Simulating);

}

private void BtnSimulate\_UpdateText(bool simulating)

{

UpdateFormText();

if (simulating)

{

btnSimulate.Text = "Stop Simulation";

}

else

{

btnSimulate.Text = "Start Simulation";

}

}

public void UpdateFormText()

{

Text = $"{builder.GetBoard().Name} - {(builder.Simulating ? "Simulating" : "Editing")}";

}

private void Builder\_SimulatingChange(bool simulating)

{

insertToolStripMenuItem.Enabled = !simulating;

editToolStripMenuItem.Enabled = !simulating;

}

}

### Builder

public class Builder : UserControl

{private ContextMenuStrip genericMenu, componentMenu;

private ToolStripLabel compNameMenuLabel;

private ToolStripMenuItem startSimMenuItem, pasteCompMenuItem, createBuiltinCompMenuItem, createBoardCompMenuItem, openSettingsMenuItem, moveCompMenuItem, copyCompMenuItem, deleteCompMenuItem;

private ToolStripSeparator compNameMenuSep, openSettingsMenuSep;

private DragType dragType = DragType.None;

private bool panning;

private Point panLastMouseLocation;

private Selection selection;

private IComponent rightClickedComp;

private Point rightClickMouseLoc;

private IComponent clipboardComp;

private IComponent dragComp;

private Point dragOffset;

private bool dragResetIsDel;

private Pos dragResetPos;

private Rotation dragResetRot;

private Point dragNewPoint;

private Rotation dragNewRot;

private Pos wireStart;

private Pos wireEnd;

protected Matrix transformationMatrix;

private Board board;

private string storedFilename;

public ColourScheme colourScheme { get; private set; }

public bool Simulating { get; private set; } = false;

private int tps = 100;

public int TPS

{

get

{

return tps;

}

set

{

tps = value;

simulationTimer.Interval = (int)Math.Round((double)(1000 / tps));

}

}

private Timer simulationTimer;

public Board GetBoard()

{

return board;

}

private static string ConstructDefaultFilename(string boardName)

{

return $"Boards/{boardName}.brd";

}

public static Builder LoadBoard(string filename)

{

return new Builder(Board.Load(filename), filename);

}

public static Builder NewBoard(string name) {

return new Builder(new Board(name));

}

public void OpenLoadBoard(string filename)

{

board = Board.Load(filename);

storedFilename = filename;

selection.Deselect();

SetSimulation(false);

Invalidate();

}

public void OpenNewBoard(string name)

{

board = new Board(name);

storedFilename = null;

selection.Deselect();

SetSimulation(false);

Invalidate();

}

public void SaveBoard(string filename = null)

{

storedFilename = filename ?? storedFilename ?? ConstructDefaultFilename(board.Name);

board.Save(storedFilename);

}

private Builder(Board board, string filename = null)

{

storedFilename = filename;

DoubleBuffered = true;

clipboardComp = null;

transformationMatrix = new Matrix();

transformationMatrix.Scale(20, 20);

colourScheme = new ColourScheme

{

Background = Color.White,

ComponentEdge = Color.Black,

ComponentBackground = Color.LightYellow,

Wire = Color.Black,

WireFloating = Color.Gray,

WireLow = Color.SkyBlue,

WirePulledLow = Color.LightSkyBlue,

WireHigh = Color.Blue,

WirePulledHigh = Color.LightBlue,

WireIllegal = Color.Red,

Grid = Color.FromArgb(63, Color.Black),

Selection = Color.Red

};

simulationTimer = new Timer

{

Interval = (int)Math.Round((double)(1000 / tps)),

Enabled = false

};

simulationTimer.Tick += SimulationTick;

this.board = board;

startSimMenuItem = new ToolStripMenuItem

{

Name = "tsmiStartSim",

Text = "Start Simulation"

};

startSimMenuItem.Click += StartSimMenuItem\_Click;

pasteCompMenuItem = new ToolStripMenuItem

{

Name = "tsmiPasteComp",

Text = "Paste Component"

};

pasteCompMenuItem.Click += PasteCompMenuItem\_Click;

createBuiltinCompMenuItem = new ToolStripMenuItem

{

Name = "tsmiCreateBuiltinComp",

Text = "Create Builtin Component"

};

createBuiltinCompMenuItem.Click += CreateBuiltinCompMenuItem\_Click;

createBoardCompMenuItem = new ToolStripMenuItem

{

Name = "tsmiCreateBoardComp",

Text = "Create Board Component"

};

createBoardCompMenuItem.Click += CreateBoardCompMenuItem\_Click;

genericMenu = new ContextMenuStrip

{

Name = "cmsGeneric"

};

genericMenu.Items.AddRange(new ToolStripItem[] { startSimMenuItem, pasteCompMenuItem, createBuiltinCompMenuItem, createBoardCompMenuItem });

compNameMenuLabel = new ToolStripLabel

{

Name = "tslCompName"

};

compNameMenuSep = new ToolStripSeparator

{

Name = "tssCompName"

};

openSettingsMenuItem = new ToolStripMenuItem

{

Name = "tsmiOpenSettings",

Text = "Open Settings"

};

openSettingsMenuItem.Click += OpenSettingsMenuItem\_Click;

openSettingsMenuSep = new ToolStripSeparator

{

Name = "tssOpenSettings"

};

moveCompMenuItem = new ToolStripMenuItem

{

Name = "tsmiMoveComp",

Text = "Move Component"

};

moveCompMenuItem.Click += MoveCompMenuItem\_Click;

copyCompMenuItem = new ToolStripMenuItem

{

Name = "tsmiCopyComp",

Text = "Copy Component"

};

copyCompMenuItem.Click += CopyCompMenuItem\_Click;

deleteCompMenuItem = new ToolStripMenuItem

{

Name = "tsmiDeleteComp",

Text = "Delete Component"

};

deleteCompMenuItem.Click += DeleteCompMenuItem\_Click;

componentMenu = new ContextMenuStrip

{

Name = "cmsSettingsComponent"

};

componentMenu.Items.AddRange(new ToolStripItem[] { moveCompMenuItem, copyCompMenuItem, deleteCompMenuItem });

}

private void StartSimMenuItem\_Click(object sender, EventArgs e)

{

SetSimulation(true);

}

private void PasteCompMenuItem\_Click(object sender, EventArgs e)

{

if (clipboardComp != null)

{

if (dragType == DragType.None)

{

StartDraggingComponent(clipboardComp.Copy(), rightClickMouseLoc, true, true);

dragType = DragType.MoveComponent;

}

}

}

private void CreateBuiltinCompMenuItem\_Click(object sender, EventArgs e)

{

CreateBuiltinComponent();

Invalidate();

}

private void CreateBoardCompMenuItem\_Click(object sender, EventArgs e)

{

CreateBoardComponent();

Invalidate();

}

private void OpenSettingsMenuItem\_Click(object sender, EventArgs e)

{

if (rightClickedComp is ISettingsComponent settingsComp) {

settingsComp.OpenSettings();

Invalidate();

}

}

private void MoveCompMenuItem\_Click(object sender, EventArgs e)

{

if (rightClickedComp != null)

{

StartDraggingComponent(rightClickedComp, rightClickMouseLoc, true);

dragType = DragType.MoveComponent;

}

}

private void CopyCompMenuItem\_Click(object sender, EventArgs e)

{

if (rightClickedComp != null)

{

clipboardComp = rightClickedComp.Copy();

}

}

private void DeleteCompMenuItem\_Click(object sender, EventArgs e)

{

if (rightClickedComp != null)

{

rightClickedComp.Remove();

Invalidate();

}

}

public void CreateBuiltinComponent()

{

ComponentSelectionForm form = new ComponentSelectionForm();

if (form.ShowDialog() == DialogResult.OK)

{

StartDraggingComponent(form.GetComponent(), new Point(), true, true);

dragType = DragType.MoveComponent;

}

}

public void CreateBoardComponent()

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.Filter = "Boards (\*.brd)|\*.brd|All files (\*.\*)|\*.\*";

openFileDialog.FilterIndex = 0;

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

Board intBoard = Board.Load(openFileDialog.FileName);

string[] overlap = intBoard.GetBoardList().Select(board => board.Name).Intersect(board.GetBoardList().Select(board => board.Name)).ToArray();

if (overlap.Count() > 0)

{

if (MessageBox.Show(

overlap.Prepend("The following boards are found within both the board you just selected and the board you are working on:").Aggregate((s1, s2) => s1 + "\n\t - " + s2) + "\nIf this might cause any issues, cancel now and double check. If this is intended, continue.",

"Overlap Detected",

MessageBoxButtons.OKCancel

) == DialogResult.Cancel)

{

return;

}

}

StartDraggingComponent(new BoardContainerComponent(intBoard, false), new Point(), true, true);

dragType = DragType.MoveComponent;

}

}

}

public void CopySelection()

{

if (selection.HasComp())

{

clipboardComp = selection.SelectedComp.Copy();

}

}

public void CutSelection()

{

if (selection.HasComp())

{

clipboardComp = selection.SelectedComp;

clipboardComp.Remove();

}

}

public void PasteSelection()

{

if (clipboardComp != null)

{

if (dragType == DragType.None)

{

StartDraggingComponent(clipboardComp.Copy(), new Point(), true, true);

dragType = DragType.MoveComponent;

}

}

}

public void SetSimulation(bool simulate)

{

if (dragType == DragType.None)

{

Simulating = simulate;

simulationTimer.Enabled = simulate;

if (Simulating)

{

selection.Deselect();

board.ResetForSimulation();

} else {

board.ResetToFloating();

}

SimulatingChange.Invoke(simulate);

Invalidate();

}

}

public event Action<bool> SimulatingChange;

private void SimulationTick(object sender, EventArgs e)

{

Dictionary<Pos, Pin.State> before = board.GetStateToCheckForChanges();

board.Tick();

Dictionary<Pos, Pin.State> after = board.GetStateToCheckForChanges();

foreach (Pos pos in before.Keys)

{

if (!after.ContainsKey(pos) || before[pos] != after[pos])

{

Invalidate();

break;

}

}

}

private Matrix GetInvertedTransformationMatrix()

{

Matrix invertedMatrix = transformationMatrix.Clone();

invertedMatrix.Invert();

return invertedMatrix;

}

private Point DetransformPoint(Point point)

{

Point[] points = new Point[] { point };

DetransformPoints(points);

return points[0];

}

private PointF DetransformPointF(PointF point)

{

PointF[] points = new PointF[] { point };

DetransformPointFs(points);

return points[0];

}

private void DetransformPoints(Point[] points)

{

GetInvertedTransformationMatrix().TransformPoints(points);

}

private void DetransformPointFs(PointF[] points)

{

GetInvertedTransformationMatrix().TransformPoints(points);

}

private IComponent GetClickedComponent(Point mouseLoc)

{

PointF mousePos = DetransformPointF(mouseLoc);

foreach (IComponent comp in board.GetComponents())

{

if (comp.GetOffsetComponentBounds().Contains(mousePos))

{

return comp;

}

}

return null;

}

private Wire GetClickedWire(Point mouseLoc)

{

PointF mousePos = DetransformPointF(mouseLoc);

foreach (Wire wire in board.GetAllWires())

{

if (wire.InflatedBounds().Contains(mousePos))

{

return wire;

}

}

return null;

}

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

Graphics graphics = e.Graphics;

graphics.ResetTransform();

graphics.Clear(Color.White);

graphics.MultiplyTransform(transformationMatrix);

Point[] corners = new Point[] { new Point(0, 0), new Point(Width - 1, Height - 1) };

DetransformPoints(corners);

board.Render(graphics, Simulating, Rectangle.FromLTRB(corners[0].X - 1, corners[0].Y - 1, corners[1].X + 1, corners[1].Y + 1), colourScheme);

if (selection.HasObject())

{

RectangleF bounds = new RectangleF();

if (selection.HasComp())

{

bounds = selection.SelectedComp.GetOffsetComponentBounds();

}

else if (selection.HasWire())

{

bounds = selection.SelectedWire.InflatedBounds();

}

graphics.DrawRectangle(new Pen(colourScheme.Selection, 0.05F), bounds.X, bounds.Y, bounds.Width, bounds.Height);

}

Matrix matrix = new Matrix();

if (dragType == DragType.MoveComponent)

{

using (new TransformRestorer(graphics))

{

matrix.Reset();

Point newPoint = DetransformPoint(dragNewPoint);

matrix.Translate(newPoint.X, newPoint.Y);

matrix.Rotate((float)dragNewRot);

graphics.MultiplyTransform(matrix);

dragComp.Render(graphics, Simulating, colourScheme);

}

//matrix.Invert();

//graphics.MultiplyTransform(matrix);

}

if (dragType == DragType.DrawWire)

{

graphics.DrawLine(new Pen(colourScheme.Wire, 0.05F), wireStart.ToPoint(), wireEnd.ToPoint());

}

graphics.ResetTransform();

graphics.DrawRectangle(new Pen(colourScheme.Selection, 1), 0, 0, Width - 1, Height - 1);

}

protected override void OnLayout(LayoutEventArgs e)

{

base.OnLayout(e);

Invalidate();

}

protected override void OnKeyDown(KeyEventArgs e)

{

base.OnKeyDown(e);

if (!Simulating)

{

if (selection.HasObject())

{

if (e.KeyCode == Keys.Delete)

{

selection.Remove();

board.SimplifyWires();

Invalidate();

}

if (selection.HasComp())

{

if (e.KeyCode == (Keys.Control | Keys.C))

{

if (dragType == DragType.None)

{

selection.Select(selection.SelectedComp.Copy());

dragType = DragType.MoveComponent;

StartDraggingComponent(selection.SelectedComp, DetransformPoint(MousePosition));

}

}

}

} else

{

if (e.KeyCode == Keys.Delete)

{

foreach (Wire wire in board.GetAllWires())

{

if (wire.Collision(Pos.FromPoint(DetransformPoint(MousePosition))))

{

wire.Remove();

}

}

}

}

}

}

private void StartDraggingComponent(IComponent comp, Point mouseLoc, bool offsetIsZero = false, bool resetIsDel = false)

{

dragComp = comp;

dragResetIsDel = resetIsDel;

dragResetPos = comp.GetComponentPos();

dragResetRot = comp.GetComponentRotation();

Point[] point = { dragResetPos.ToPoint() };

transformationMatrix.TransformPoints(point);

dragOffset = offsetIsZero ? new Point(0, 0) : new Point(point[0].X - mouseLoc.X, point[0].Y - mouseLoc.Y);

dragNewRot = comp.GetComponentRotation();

dragNewPoint = new Point(mouseLoc.X + dragOffset.X, mouseLoc.Y + dragOffset.Y);

comp.Remove();

Invalidate();

}

private void PutDownDraggedComponent()

{

Point newPoint = DetransformPoint(dragNewPoint);

Pos newPos = Pos.FromPoint(newPoint);

Matrix matrix = new Matrix();

matrix.Rotate((float)dragComp.GetComponentRotation());

matrix.Translate(newPos.X, newPos.Y);

RectangleF bounds = dragComp.GetComponentBounds();

PointF[] corners = { new PointF(bounds.Left, bounds.Top), new PointF(bounds.Right, bounds.Bottom) };

matrix.TransformPoints(corners);

bounds = RectangleF.FromLTRB(corners[0].X, corners[0].Y, corners[1].X, corners[1].Y);

if (board.CheckAllowed(bounds))

{

dragComp.Place(newPos, dragNewRot, board);

dragComp = null;

Invalidate();

} else

{

ResetDraggedComponent();

}

}

private void ResetDraggedComponent()

{

if (!dragResetIsDel)

{

dragComp.Place(dragResetPos, dragResetRot, board);

}

dragComp = null;

Invalidate();

}

private void StartWire(Pos pos)

{

board.SimplifyWires();

Rectangle bounds;

HashSet<Wire> removeWires = new HashSet<Wire>();

HashSet<(Pos, Pos)> addWires = new HashSet<(Pos, Pos)>();

foreach (Wire wire in board.GetAllWires())

{

bounds = wire.Bounds();

if (wire.Collision(pos))

{

removeWires.Add(wire);

addWires.Add((wire.Pos1, pos));

addWires.Add((wire.Pos2, pos));

}

}

foreach (Wire wire in removeWires)

{

wire.Remove();

}

foreach ((Pos, Pos) wire in addWires)

{

new Wire(wire.Item1, wire.Item2, board);

}

wireStart = pos;

}

protected override void OnMouseClick(MouseEventArgs e)

{

base.OnMouseClick(e);

if (e.Button == MouseButtons.Left)

{

if (Simulating)

{

IComponent clickedComp = GetClickedComponent(e.Location);

if (clickedComp != null)

{

if (clickedComp is IInteractibleComponent interactComp)

{

interactComp.Interact();

}

}

} else {

if (dragType == DragType.MoveComponent)

{

PutDownDraggedComponent();

dragType = DragType.None;

} else if (dragType == DragType.DrawWire)

{

if (wireStart != wireEnd)

{

new Wire(wireStart, wireEnd, board);

}

StartWire(wireEnd);

} else

{

IComponent clickedComp = GetClickedComponent(e.Location);

Wire clickedWire = GetClickedWire(e.Location);

Pos clickedPos = Pos.FromPoint(DetransformPoint(e.Location));

if (board.GetComponents().Length != 0 && board.GetComponents().Select(comp => comp.GetAllPinPositions()).Aggregate((posArr1, posArr2) => posArr1.Concat(posArr2).ToArray()).Contains(clickedPos))

{

StartWire(clickedPos);

dragType = DragType.DrawWire;

} else if (clickedComp != null)

{

if (clickedComp == selection.SelectedComp)

{

StartDraggingComponent(clickedComp, e.Location);

dragType = DragType.MoveComponent;

} else

{

selection.Select(clickedComp);

}

} else if (clickedWire != null)

{

if (clickedWire == selection.SelectedWire)

{

StartWire(clickedPos);

dragType = DragType.DrawWire;

selection.Deselect();

} else

{

selection.Select(clickedWire);

}

} else

{

selection.Deselect();

}

}

}

} else if (e.Button == MouseButtons.Right)

{

if (!Simulating)

{

IComponent clickedComp = GetClickedComponent(e.Location);

ContextMenuStrip contextMenuStrip;

if (clickedComp != null)

{

selection.Select(clickedComp);

contextMenuStrip = componentMenu;

if (clickedComp is ISettingsComponent)

{

contextMenuStrip.Items.Insert(0, openSettingsMenuItem);

contextMenuStrip.Items.Insert(1, openSettingsMenuSep);

} else

{

contextMenuStrip.Items.Remove(openSettingsMenuItem);

contextMenuStrip.Items.Remove(openSettingsMenuSep);

}

if (clickedComp is IBoardInterfaceComponent interfaceComp)

{

compNameMenuLabel.Text = interfaceComp.GetComponentName();

contextMenuStrip.Items.Insert(0, compNameMenuLabel);

contextMenuStrip.Items.Insert(1, compNameMenuSep);

} else if (clickedComp is IBoardContainerComponent contComp)

{

compNameMenuLabel.Text = contComp.GetInternalBoard().Name;

contextMenuStrip.Items.Insert(0, compNameMenuLabel);

contextMenuStrip.Items.Insert(1, compNameMenuSep);

} else

{

contextMenuStrip.Items.Remove(compNameMenuLabel);

contextMenuStrip.Items.Remove(compNameMenuSep);

}

}

else

{

contextMenuStrip = genericMenu;

}

rightClickedComp = clickedComp;

rightClickMouseLoc = e.Location;

contextMenuStrip.Show(Cursor.Position);

}

}

Invalidate();

}

protected override void OnMouseDoubleClick(MouseEventArgs e)

{

base.OnMouseDoubleClick(e);

if (e.Button == MouseButtons.Left)

{

if (!Simulating)

{

if (dragType == DragType.MoveComponent)

{

ResetDraggedComponent();

dragType = DragType.None;

}

if (dragType == DragType.DrawWire)

{

dragType = DragType.None;

} else

{

IComponent clickedComp = GetClickedComponent(e.Location);

if (clickedComp != null && clickedComp == selection.SelectedComp)

{

if (clickedComp is ISettingsComponent settingsComp)

{

settingsComp.OpenSettings();

}

else

{

StartDraggingComponent(clickedComp, e.Location);

dragType = DragType.MoveComponent;

}

}

else

{

selection.Select(clickedComp);

}

}

}

}

Invalidate();

}

protected override void OnMouseDown(MouseEventArgs e)

{

base.OnMouseDown(e);

if (e.Button == MouseButtons.Middle)

{

if (!panning)

{

panning = true;

panLastMouseLocation = e.Location;

}

}

}

protected override void OnMouseUp(MouseEventArgs e)

{

base.OnMouseUp(e);

if (e.Button == MouseButtons.Middle)

{

if (panning)

{

panning = false;

}

}

}

protected override void OnMouseMove(MouseEventArgs e)

{

base.OnMouseMove(e);

if (panning)

{

PointF[] locs = new PointF[] { e.Location, panLastMouseLocation };

DetransformPointFs(locs);

transformationMatrix.Translate(locs[0].X - locs[1].X, locs[0].Y - locs[1].Y);

panLastMouseLocation = e.Location;

Invalidate();

}

if (dragType == DragType.MoveComponent)

{

dragNewPoint = new Point(e.Location.X + dragOffset.X, e.Location.Y + dragOffset.Y);

Invalidate();

}

if (dragType == DragType.DrawWire)

{

wireEnd = Pos.FromPoint(DetransformPoint(e.Location));

if (wireStart.X != wireEnd.X && wireStart.Y != wireEnd.Y)

{

if (Math.Abs(wireEnd.X - wireStart.X) > Math.Abs(wireEnd.Y - wireStart.Y))

{

wireEnd = new Pos(wireEnd.X, wireStart.Y);

} else

{

wireEnd = new Pos(wireStart.X, wireEnd.Y);

}

}

Invalidate();

}

}

protected override void OnMouseWheel(MouseEventArgs e)

{

base.OnMouseWheel(e);

if (dragType == DragType.MoveComponent)

{

dragNewRot = dragNewRot.AddRotation(e.Delta > 0 ? Rotation.CLOCKWISE : Rotation.ANTICLOCKWISE);

Invalidate();

} else

{

float scale = (Math.Sign(e.Delta) \* 0.1F) + 1;

PointF loc = DetransformPointF(e.Location);

transformationMatrix.Translate(loc.X, loc.Y);

transformationMatrix.Scale(scale, scale);

float[] elements = transformationMatrix.Elements;

transformationMatrix.Scale(

Math.Max(Math.Min(elements[0], 100F), 10F) / elements[0],

Math.Max(Math.Min(elements[3], 100F), 10F) / elements[3]

);

transformationMatrix.Translate(-loc.X, -loc.Y);

Invalidate();

}

}

}

#### DragType

private enum DragType

{

None, MoveComponent, DrawWire

}

#### Selection

private struct Selection

{

public ComponentOrWire SelectedObject;

public IComponent SelectedComp { get { return SelectedObject.Comp(); } }

public Wire SelectedWire { get { return SelectedObject.Wire(); } }

public bool HasObject() { return SelectedObject.Exists(); }

public bool HasComp() { return SelectedObject.IsComp(); }

public bool HasWire() { return SelectedObject.IsWire(); }

public void Deselect() { SelectedObject.Reset(); }

public void Select(IComponent comp) { SelectedObject.Set(comp); }

public void Select(Wire wire) { SelectedObject.Set(wire); }

public void Remove()

{

if (SelectedObject.IsComp())

{

SelectedObject.Comp().Remove();

} else if (SelectedObject.IsWire())

{

SelectedObject.Wire().Remove();

}

SelectedObject.Reset();

}

}

##### ComponentOrWire

public struct ComponentOrWire

{

private IComponent comp;

private Wire wire;

public bool Exists() { return IsComp() || IsWire(); }

public bool IsComp() { return comp != null; }

public bool IsWire() { return wire != null; }

public IComponent Comp() { return comp; }

public Wire Wire() { return wire; }

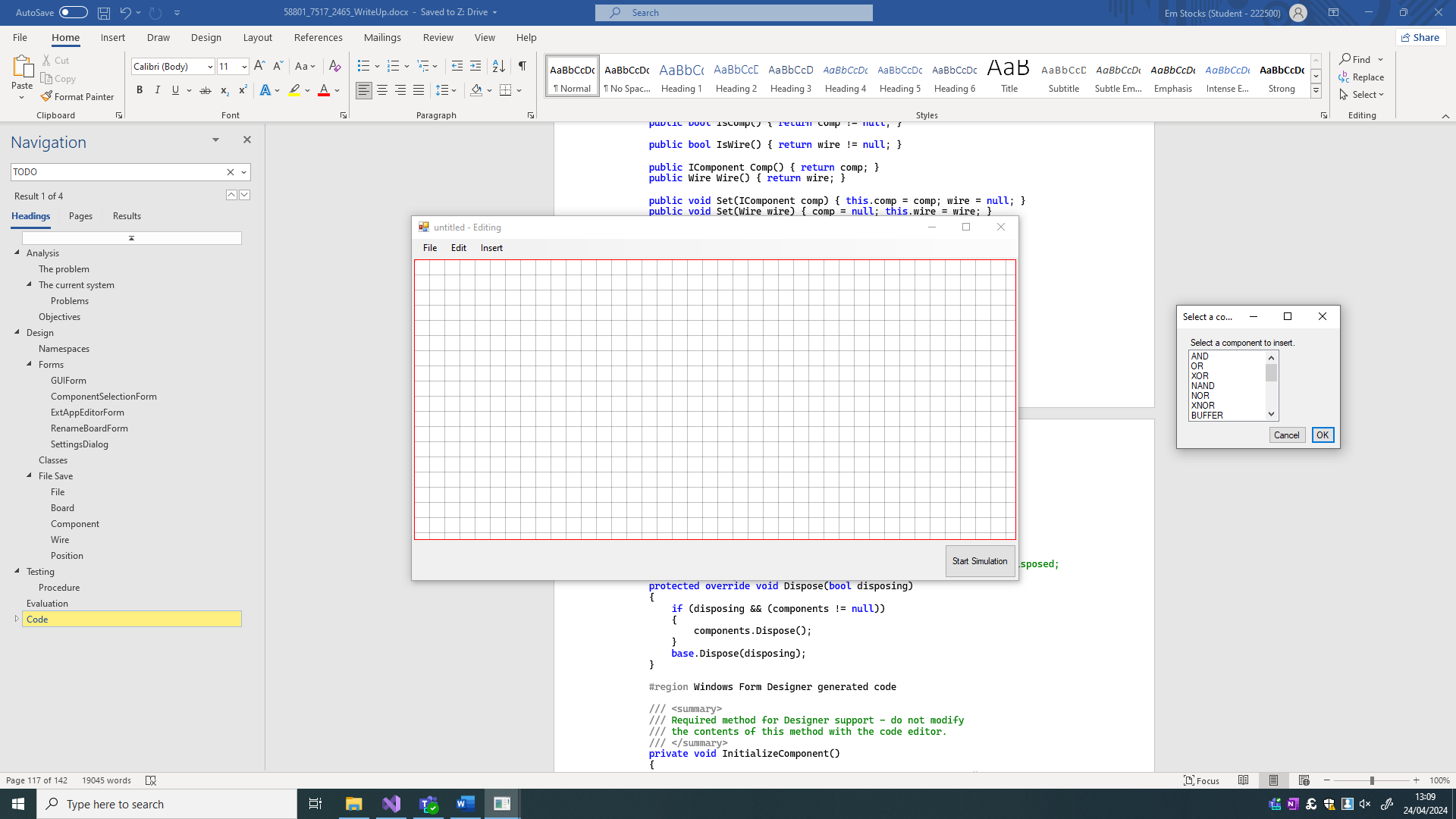
public void Set(IComponent comp) { this.comp = comp; wire = null; }

public void Set(Wire wire) { comp = null; this.wire = wire; }

public void Reset() { comp = null; wire = null; }

}

### ComponentSelectionForm



partial class ComponentSelectionForm

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.lstComponents = new System.Windows.Forms.ListBox();

this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();

this.lblInstruction = new System.Windows.Forms.Label();

this.btnOK = new System.Windows.Forms.Button();

this.btnCancel = new System.Windows.Forms.Button();

this.tableLayoutPanel1.SuspendLayout();

this.SuspendLayout();

//

// lstComponents

//

this.tableLayoutPanel1.SetColumnSpan(this.lstComponents, 3);

this.lstComponents.FormattingEnabled = true;

this.lstComponents.Location = new System.Drawing.Point(3, 16);

this.lstComponents.Name = "lstComponents";

this.lstComponents.Size = new System.Drawing.Size(120, 95);

this.lstComponents.TabIndex = 0;

//

// tableLayoutPanel1

//

this.tableLayoutPanel1.AutoSize = true;

this.tableLayoutPanel1.ColumnCount = 3;

this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 100F));

this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tableLayoutPanel1.Controls.Add(this.lstComponents, 0, 1);

this.tableLayoutPanel1.Controls.Add(this.lblInstruction, 0, 0);

this.tableLayoutPanel1.Controls.Add(this.btnOK, 2, 2);

this.tableLayoutPanel1.Controls.Add(this.btnCancel, 1, 2);

this.tableLayoutPanel1.Location = new System.Drawing.Point(12, 12);

this.tableLayoutPanel1.Name = "tableLayoutPanel1";

this.tableLayoutPanel1.RowCount = 3;

this.tableLayoutPanel1.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tableLayoutPanel1.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tableLayoutPanel1.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tableLayoutPanel1.Size = new System.Drawing.Size(200, 143);

this.tableLayoutPanel1.TabIndex = 1;

//

// lblInstruction

//

this.lblInstruction.AutoSize = true;

this.tableLayoutPanel1.SetColumnSpan(this.lblInstruction, 3);

this.lblInstruction.Location = new System.Drawing.Point(3, 0);

this.lblInstruction.Name = "lblInstruction";

this.lblInstruction.Size = new System.Drawing.Size(145, 13);

this.lblInstruction.TabIndex = 1;

this.lblInstruction.Text = "Select a component to insert.";

//

// btnOK

//

this.btnOK.AutoSize = true;

this.btnOK.DialogResult = System.Windows.Forms.DialogResult.OK;

this.btnOK.Location = new System.Drawing.Point(165, 117);

this.btnOK.Name = "btnOK";

this.btnOK.Size = new System.Drawing.Size(32, 23);

this.btnOK.TabIndex = 2;

this.btnOK.Text = "OK";

this.btnOK.UseVisualStyleBackColor = true;

//

// btnCancel

//

this.btnCancel.AutoSize = true;

this.btnCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;

this.btnCancel.Location = new System.Drawing.Point(109, 117);

this.btnCancel.Name = "btnCancel";

this.btnCancel.Size = new System.Drawing.Size(50, 23);

this.btnCancel.TabIndex = 3;

this.btnCancel.Text = "Cancel";

this.btnCancel.UseVisualStyleBackColor = true;

//

// ComponentSelectionForm

//

this.AcceptButton = this.btnOK;

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.AutoSize = true;

this.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.CancelButton = this.btnCancel;

this.ClientSize = new System.Drawing.Size(800, 450);

this.Controls.Add(this.tableLayoutPanel1);

this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;

this.Name = "ComponentSelectionForm";

this.Text = "Select a component";

this.tableLayoutPanel1.ResumeLayout(false);

this.tableLayoutPanel1.PerformLayout();

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.ListBox lstComponents;

private System.Windows.Forms.TableLayoutPanel tableLayoutPanel1;

private System.Windows.Forms.Label lblInstruction;

private System.Windows.Forms.Button btnOK;

private System.Windows.Forms.Button btnCancel;

}

public partial class ComponentSelectionForm : Form

{

public ComponentSelectionForm()

{

InitializeComponent();

lstComponents.Items.AddRange(ReadWriteImplementation.Constructors.Keys.Where(id => id != "BOARD").ToArray());

}

public IComponent GetComponent()

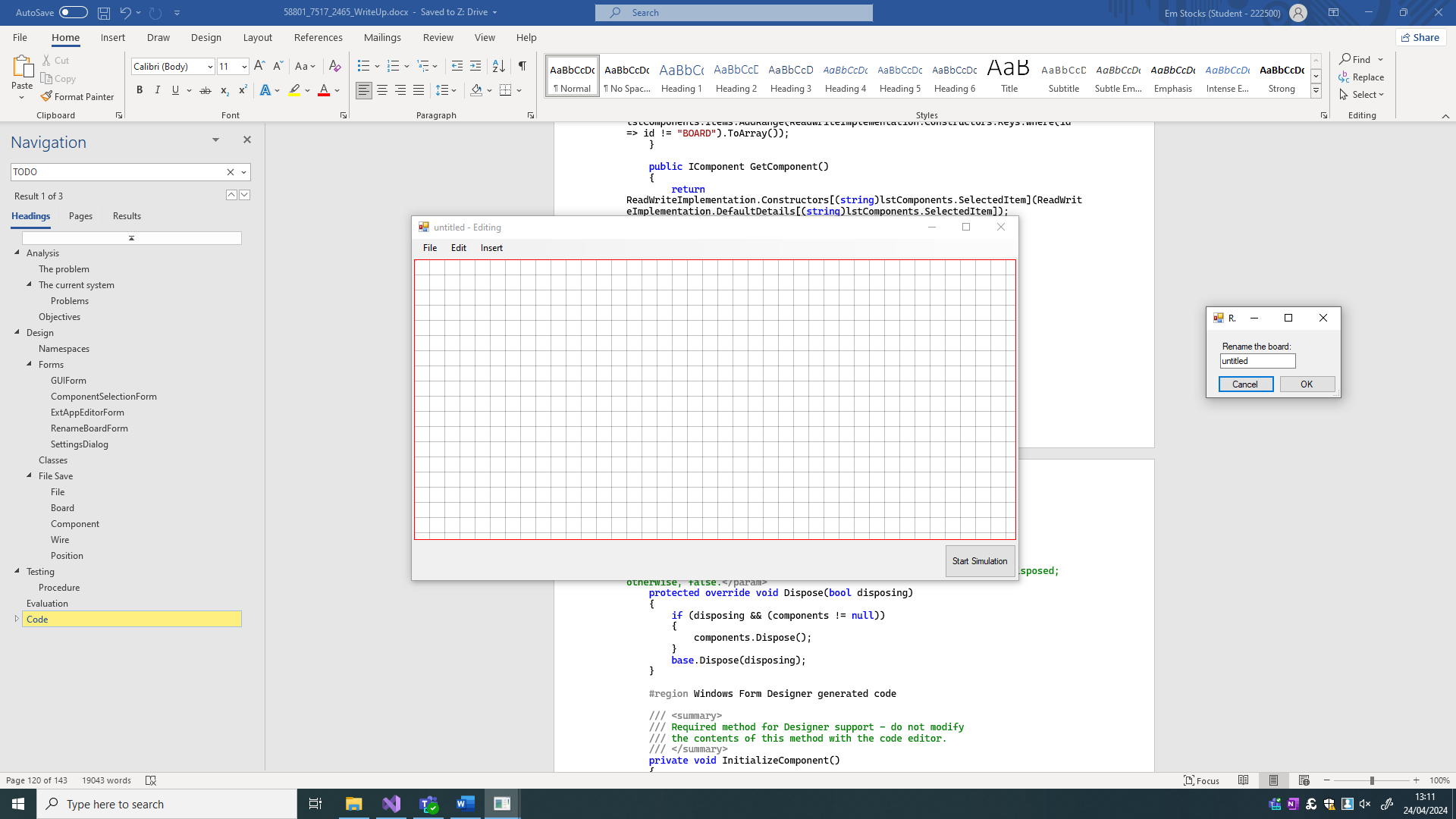
{

return ReadWriteImplementation.Constructors[(string)lstComponents.SelectedItem](ReadWriteImplementation.DefaultDetails[(string)lstComponents.SelectedItem]);

}

}

### RenameBoardForm



partial class RenameBoardForm

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();

this.btnCancel = new System.Windows.Forms.Button();

this.btnOK = new System.Windows.Forms.Button();

this.tableLayoutPanel2 = new System.Windows.Forms.TableLayoutPanel();

this.txtName = new System.Windows.Forms.TextBox();

this.lblInstruction = new System.Windows.Forms.Label();

this.tableLayoutPanel1.SuspendLayout();

this.tableLayoutPanel2.SuspendLayout();

this.SuspendLayout();

//

// tableLayoutPanel1

//

this.tableLayoutPanel1.AutoSize = true;

this.tableLayoutPanel1.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.tableLayoutPanel1.ColumnCount = 3;

this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tableLayoutPanel1.Controls.Add(this.btnCancel, 1, 1);

this.tableLayoutPanel1.Controls.Add(this.btnOK, 2, 1);

this.tableLayoutPanel1.Controls.Add(this.tableLayoutPanel2, 0, 0);

this.tableLayoutPanel1.Location = new System.Drawing.Point(12, 12);

this.tableLayoutPanel1.Name = "tableLayoutPanel1";

this.tableLayoutPanel1.RowCount = 2;

this.tableLayoutPanel1.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tableLayoutPanel1.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tableLayoutPanel1.Size = new System.Drawing.Size(162, 74);

this.tableLayoutPanel1.TabIndex = 0;

//

// btnCancel

//

this.btnCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;

this.btnCancel.Location = new System.Drawing.Point(3, 48);

this.btnCancel.Name = "btnCancel";

this.btnCancel.Size = new System.Drawing.Size(75, 23);

this.btnCancel.TabIndex = 0;

this.btnCancel.Text = "Cancel";

this.btnCancel.UseVisualStyleBackColor = true;

//

// btnOK

//

this.btnOK.DialogResult = System.Windows.Forms.DialogResult.OK;

this.btnOK.Location = new System.Drawing.Point(84, 48);

this.btnOK.Name = "btnOK";

this.btnOK.Size = new System.Drawing.Size(75, 23);

this.btnOK.TabIndex = 1;

this.btnOK.Text = "OK";

this.btnOK.UseVisualStyleBackColor = true;

//

// tableLayoutPanel2

//

this.tableLayoutPanel2.AutoSize = true;

this.tableLayoutPanel2.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.tableLayoutPanel2.ColumnCount = 1;

this.tableLayoutPanel1.SetColumnSpan(this.tableLayoutPanel2, 3);

this.tableLayoutPanel2.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tableLayoutPanel2.Controls.Add(this.txtName, 0, 1);

this.tableLayoutPanel2.Controls.Add(this.lblInstruction, 0, 0);

this.tableLayoutPanel2.Location = new System.Drawing.Point(3, 3);

this.tableLayoutPanel2.Name = "tableLayoutPanel2";

this.tableLayoutPanel2.RowCount = 2;

this.tableLayoutPanel2.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tableLayoutPanel2.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tableLayoutPanel2.Size = new System.Drawing.Size(106, 39);

this.tableLayoutPanel2.TabIndex = 2;

//

// txtName

//

this.txtName.Location = new System.Drawing.Point(3, 16);

this.txtName.Name = "txtName";

this.txtName.Size = new System.Drawing.Size(100, 20);

this.txtName.TabIndex = 1;

//

// lblInstruction

//

this.lblInstruction.AutoSize = true;

this.lblInstruction.Location = new System.Drawing.Point(3, 0);

this.lblInstruction.Name = "lblInstruction";

this.lblInstruction.Size = new System.Drawing.Size(98, 13);

this.lblInstruction.TabIndex = 2;

this.lblInstruction.Text = "Rename the board:";

//

// RenameBoardForm

//

this.AcceptButton = this.btnOK;

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.AutoSize = true;

this.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.CancelButton = this.btnCancel;

this.ClientSize = new System.Drawing.Size(800, 450);

this.Controls.Add(this.tableLayoutPanel1);

this.Name = "RenameBoardForm";

this.Text = "Rename Board";

this.Load += new System.EventHandler(this.RenameBoardForm\_Load);

this.tableLayoutPanel1.ResumeLayout(false);

this.tableLayoutPanel1.PerformLayout();

this.tableLayoutPanel2.ResumeLayout(false);

this.tableLayoutPanel2.PerformLayout();

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.TableLayoutPanel tableLayoutPanel1;

private System.Windows.Forms.Button btnCancel;

private System.Windows.Forms.Button btnOK;

private System.Windows.Forms.TableLayoutPanel tableLayoutPanel2;

private System.Windows.Forms.TextBox txtName;

private System.Windows.Forms.Label lblInstruction;

}

public partial class RenameBoardForm : Form

{

private string BoardName;

public RenameBoardForm(string name)

{

BoardName = name;

InitializeComponent();

}

private void RenameBoardForm\_Load(object sender, EventArgs e)

{

txtName.Text = BoardName;

}

public string NewBoardName()

{

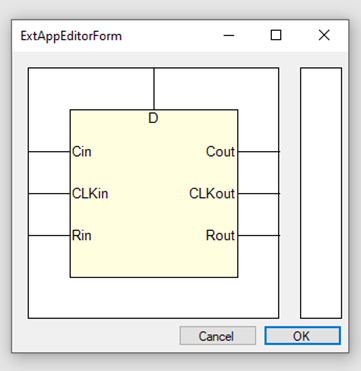
return txtName.Text;

}

}

### ExtApp

#### ExtAppEditorForm



partial class ExtAppEditorForm

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

//this.extAppEditor1 = new CircuitMaker.GUI.ExtApp.ExtAppEditor();

this.tblMain = new System.Windows.Forms.TableLayoutPanel();

this.btnCancel = new System.Windows.Forms.Button();

this.btnOK = new System.Windows.Forms.Button();

this.tblMain.SuspendLayout();

this.SuspendLayout();

//

// extAppEditor

//

this.tblMain.SetColumnSpan(this.extAppEditor, 3);

this.extAppEditor.Location = new System.Drawing.Point(3, 3);

this.extAppEditor.Name = "extAppEditor";

this.extAppEditor.TabIndex = 0;

//

// tblMain

//

this.tblMain.AutoSize = true;

this.tblMain.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.tblMain.ColumnCount = 3;

this.tblMain.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 100F));

this.tblMain.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tblMain.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());

this.tblMain.Controls.Add(this.extAppEditor, 0, 0);

this.tblMain.Controls.Add(this.btnCancel, 1, 1);

this.tblMain.Controls.Add(this.btnOK, 2, 1);

this.tblMain.Location = new System.Drawing.Point(12, 12);

this.tblMain.Name = "tblMain";

this.tblMain.RowCount = 2;

this.tblMain.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 100F));

this.tblMain.RowStyles.Add(new System.Windows.Forms.RowStyle());

this.tblMain.TabIndex = 1;

//

// btnCancel

//

this.btnCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;

this.btnCancel.Location = new System.Drawing.Point(3, 53);

this.btnCancel.Name = "btnCancel";

this.btnCancel.Size = new System.Drawing.Size(75, 20);

this.btnCancel.TabIndex = 1;

this.btnCancel.Text = "Cancel";

this.btnCancel.UseVisualStyleBackColor = true;

//

// btnOK

//

this.btnOK.DialogResult = System.Windows.Forms.DialogResult.OK;

this.btnOK.Location = new System.Drawing.Point(84, 53);

this.btnOK.Name = "btnOK";

this.btnOK.Size = new System.Drawing.Size(75, 20);

this.btnOK.TabIndex = 2;

this.btnOK.Text = "OK";

this.btnOK.UseVisualStyleBackColor = true;

//

// ExtAppEditorForm

//

this.AcceptButton = this.btnOK;

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.AutoSize = true;

this.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.CancelButton = this.btnCancel;

this.ClientSize = new System.Drawing.Size(800, 450);

this.Controls.Add(this.tblMain);

this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;

this.Name = "ExtAppEditorForm";

this.Text = "ExtAppEditorForm";

this.tblMain.ResumeLayout(false);

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private ExtAppEditor extAppEditor;

private System.Windows.Forms.TableLayoutPanel tblMain;

private System.Windows.Forms.Button btnCancel;

private System.Windows.Forms.Button btnOK;

}

partial class ExtAppEditorForm : Form

{

public ExtAppEditorForm(IBoardContainerComponent boardContainerComp, ColourScheme colourScheme)

{

extAppEditor = new ExtAppEditor(boardContainerComp, colourScheme);

DoubleBuffered = true;

InitializeComponent();

}

public void SaveChanges()

{

extAppEditor.SaveChanges();

}

public void ResetChanges()

{

extAppEditor.ResetChanges();

}

}

#### ExtAppEditor

partial class ExtAppEditor

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Component Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.SuspendLayout();

//

// ExtAppEditor

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.Name = "ExtAppEditor";

this.ResumeLayout(false);

}

#endregion

}

public partial class ExtAppEditor : UserControl

{

private IBoardContainerComponent boardContainerComp;

public ColourScheme colourScheme;

private Dictionary<string, Board.InterfaceLocation> interfaceLocSave;

private Dictionary<IGraphicalComponent, Point?> graphicalLocSave;

private Size sizeSave;

private int scale = 40;

private int resizeStartRange = 10;

private Matrix compTransformationMatrix = new Matrix();

private Matrix graphicalsTransformationMatrix = new Matrix();

private DragState dragState;

private Point mouseDragLoc;

Rectangle compDisplayBounds;

Rectangle graphicalsDisplayBounds;

public ExtAppEditor(IBoardContainerComponent boardContainerComp, ColourScheme colourScheme)

{

DoubleBuffered = true;

InitializeComponent();

dragState = new DragState();

this.colourScheme = colourScheme;

this.boardContainerComp = boardContainerComp;

ResetSize();

SaveChanges();

}

private void ResetSize()

{

Size size = boardContainerComp.GetInternalBoard().ExternalSize;

size.Width += 2;

size.Height += 2;

float graphicalsStart = size.Width;

size.Width \*= scale;

size.Height \*= scale;

compDisplayBounds = new Rectangle(Point.Empty, size);

size.Width += scale / 2;

graphicalsStart += 0.5F;

int graphicalsWidth = GetUnplacedGraphicalComponents().Select(comp => (int)Math.Ceiling(comp.GetScaledGraphicalElementBounds().Width \* scale)).Append(scale).Aggregate(Math.Max);

graphicalsDisplayBounds = new Rectangle(size.Width, 0, graphicalsWidth, size.Height);

size.Width += graphicalsWidth;

Size = size;

Rectangle compRect = boardContainerComp.GetShape();

compTransformationMatrix.Reset();

compTransformationMatrix.Scale(scale, scale);

compTransformationMatrix.Translate(-compRect.X, -compRect.Y);

compTransformationMatrix.Translate(1, 1);

graphicalsTransformationMatrix.Reset();

graphicalsTransformationMatrix.Scale(scale, scale);

graphicalsTransformationMatrix.Translate(graphicalsStart, 0);

}

public void SaveChanges()

{

Board internalBoard = boardContainerComp.GetInternalBoard();

interfaceLocSave = new Dictionary<string, Board.InterfaceLocation>();

foreach (IBoardInterfaceComponent interfaceComp in internalBoard.GetInterfaceComponents())

{

interfaceLocSave.Add(interfaceComp.GetComponentName(), interfaceComp.GetInterfaceLocation());

}

graphicalLocSave = new Dictionary<IGraphicalComponent, Point?>();

foreach (IGraphicalComponent graphicalComp in internalBoard.GetGraphicalComponents())

{

graphicalLocSave.Add(graphicalComp, graphicalComp.GetGraphicalElementLocation());

}

sizeSave = internalBoard.ExternalSize;

}

public void ResetChanges()

{

Board internalBoard = boardContainerComp.GetInternalBoard();

foreach (string compName in interfaceLocSave.Keys)

{

internalBoard.GetInterfaceComponent(compName).SetInterfaceLocation(interfaceLocSave[compName]);

}

foreach (IGraphicalComponent graphicalComp in graphicalLocSave.Keys)

{

graphicalComp.SetGraphicalElementLocation(graphicalLocSave[graphicalComp]);

}

internalBoard.ExternalSize = sizeSave;

}

private Matrix GetInvertedCompTransformationMatrix()

{

Matrix invertedMatrix = compTransformationMatrix.Clone();

invertedMatrix.Invert();

return invertedMatrix;

}

private Point CompDetransformPoint(Point point)

{

Point[] points = new Point[] { point };

CompDetransformPoints(points);

return points[0];

}

private PointF CompDetransformPointF(PointF point)

{

PointF[] points = new PointF[] { point };

CompDetransformPointFs(points);

return points[0];

}

private void CompDetransformPoints(Point[] points)

{

GetInvertedCompTransformationMatrix().TransformPoints(points);

}

private void CompDetransformPointFs(PointF[] points)

{

GetInvertedCompTransformationMatrix().TransformPoints(points);

}

private IGraphicalComponent DetectUnplacedGraphicalClick(Point point)

{

int offset = 0;

Matrix matrix = new Matrix();

RectangleF boundsF;

Rectangle bounds;

PointF[] points;

foreach (IGraphicalComponent graphicalComp in GetUnplacedGraphicalComponents().Where(comp => comp.HasGraphics()))

{

matrix.Reset();

matrix.Translate(0, offset);

boundsF = graphicalComp.GetScaledGraphicalElementBounds();

bounds = new Rectangle((int)Math.Floor(boundsF.X \* scale), (int)Math.Floor(boundsF.Y \* scale), (int)Math.Ceiling(boundsF.Width \* scale), (int)Math.Ceiling(boundsF.Height \* scale));

offset += bounds.Height;

matrix.Translate(-bounds.X, -bounds.Y);

matrix.Multiply(graphicalsTransformationMatrix);

matrix.Invert();

points = new PointF[] { point };

matrix.TransformPoints(points);

if (boundsF.Contains(points[0]))

{

return graphicalComp;

}

}

return null;

}

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

Graphics graphics = e.Graphics;

graphics.ResetTransform();

graphics.Clear(DefaultBackColor);

Rectangle compRect = compDisplayBounds;

compRect.Width -= 1;

compRect.Height -= 1;

graphics.FillRectangle(new SolidBrush(Color.White), compRect);

graphics.DrawRectangle(new Pen(Brushes.Black, 0.5F), compRect);

Rectangle graphicalsRect = graphicalsDisplayBounds;

graphicalsRect.Width -= 1;

graphicalsRect.Height -= 1;

graphics.FillRectangle(new SolidBrush(Color.White), graphicalsRect);

graphics.DrawRectangle(new Pen(Brushes.Black, 0.5F), graphicalsRect);

graphics.MultiplyTransform(compTransformationMatrix);

boardContainerComp.Render(graphics, false, colourScheme);

boardContainerComp.RenderGraphicalElement(graphics, false, colourScheme);

graphics.ResetTransform();

graphics.MultiplyTransform(graphicalsTransformationMatrix);

int offset = 0;

Matrix matrix = new Matrix();

RectangleF boundsF;

Rectangle bounds;

float compScale;

foreach (IGraphicalComponent graphicalComp in GetUnplacedGraphicalComponents())

{

using (new TransformRestorer(graphics))

{

matrix.Reset();

matrix.Translate(0, offset);

boundsF = graphicalComp.GetScaledGraphicalElementBounds();

bounds = new Rectangle((int)Math.Floor(boundsF.X \* scale), (int)Math.Floor(boundsF.Y \* scale), (int)Math.Ceiling(boundsF.Width \* scale), (int)Math.Ceiling(boundsF.Height \* scale));

compScale = graphicalComp.GetGraphicalElementScale();

offset += bounds.Height;

matrix.Translate(-bounds.X, -bounds.Y);

matrix.Scale(compScale, compScale);

graphics.MultiplyTransform(matrix, MatrixOrder.Append);

if (dragState.IsGraphicalComp() && dragState.GetGraphicalComp() == graphicalComp)

{

graphics.FillRectangle(new HatchBrush(HatchStyle.ForwardDiagonal, colourScheme.Selection, Color.Transparent), boundsF);

}

graphicalComp.RenderGraphicalElement(graphics, false, colourScheme);

}

}

}

private IGraphicalComponent[] GetGraphicalComponents()

{

return boardContainerComp.GetInternalBoard().GetGraphicalComponents();

}

private IGraphicalComponent[] GetUnplacedGraphicalComponents()

{

return GetGraphicalComponents().Where(comp => comp.HasGraphics() && !comp.GetGraphicalElementLocation().HasValue).ToArray();

}

private IBoardInterfaceComponent[] GetInterfaceComponents()

{

return boardContainerComp.GetInternalBoard().GetInterfaceComponents();

}

private IBoardInterfaceComponent GetInterfaceComponent(string name)

{

return boardContainerComp.GetInternalBoard().GetInterfaceComponent(name);

}

private IBoardInterfaceComponent GetInterfaceComponent(Board.InterfaceLocation loc)

{

IBoardInterfaceComponent[] comps = GetInterfaceComponents().Where(comp => comp.GetInterfaceLocation() == loc).ToArray();

if (comps.Length == 0)

{

return null;

}

return comps[0];

}

protected override void OnMouseDown(MouseEventArgs e)

{

base.OnMouseDown(e);

Point farCorner = new Point(compDisplayBounds.Width - scale, compDisplayBounds.Height - scale);

if (e.Location.X > compDisplayBounds.Width)

{

IGraphicalComponent comp = DetectUnplacedGraphicalClick(e.Location);

if (comp != null)

{

dragState.SelectGraphicalComp(comp, null);

}

} else if (e.Location.X > farCorner.X - resizeStartRange && e.Location.X < farCorner.X + resizeStartRange && e.Location.Y > farCorner.Y - resizeStartRange && e.Location.Y < farCorner.Y + resizeStartRange)

{

dragState.SelectSize(boardContainerComp.GetInternalBoard().ExternalSize);

} else

{

bool onLeft = e.Location.X < scale, onRight = e.Location.X > farCorner.X,

onTop = e.Location.Y < scale, onBottom = e.Location.Y > farCorner.Y,

onLeftRight = onLeft || onRight, onTopBottom = onTop || onBottom;

if (!(onLeftRight || onTopBottom))

{

RectangleF? bounds;

foreach (IGraphicalComponent graphicalComp in GetGraphicalComponents())

{

bounds = graphicalComp.GetOffsetScaledGraphicalElementBounds();

if (bounds.HasValue && bounds.Value.Contains(CompDetransformPointF(e.Location)))

{

dragState.SelectGraphicalComp(graphicalComp, graphicalComp.GetGraphicalElementLocation());

break;

}

}

}

else if (onLeftRight ^ onTopBottom)

{

Board.InterfaceLocation.SideEnum side = ((Board.InterfaceLocation.SideEnum)0).WithLeftRightAs(onLeftRight).WithBottomRightAs(onBottom || onRight);

Point clickedPoint = CompDetransformPoint(e.Location);

Point offset = boardContainerComp.GetShape().Location;

foreach (IBoardInterfaceComponent interfaceComp in GetInterfaceComponents().Where(interfaceComp => interfaceComp.GetInterfaceLocation().Side == side))

{

if ((onLeftRight ? clickedPoint.Y - offset.Y : clickedPoint.X - offset.X) == interfaceComp.GetInterfaceLocation().Distance)

{

dragState.SelectInterfaceLoc(interfaceComp.GetComponentName(), interfaceComp.GetInterfaceLocation());

}

}

}

}

}

protected override void OnMouseUp(MouseEventArgs e)

{

base.OnMouseUp(e);

if (dragState.IsAnything())

{

dragState.Reset();

ResetSize();

Invalidate();

}

}

protected override void OnMouseMove(MouseEventArgs e)

{

base.OnMouseMove(e);

if (dragState.IsAnything())

{

mouseDragLoc = e.Location;

Point mousePoint = CompDetransformPoint(mouseDragLoc);

Board internalBoard = boardContainerComp.GetInternalBoard();

Rectangle shape = boardContainerComp.GetShape();

if (dragState.IsSize())

{

float[] bounds = new float[] { 1, 1 };

int idx;

foreach (IBoardInterfaceComponent comp in GetInterfaceComponents())

{

idx = comp.GetInterfaceLocation().Side.IsLeftRight() ? 1 : 0;

bounds[idx] = Math.Max(comp.GetInterfaceLocation().Distance + 1, bounds[idx]);

}

RectangleF? possCompBounds;

RectangleF compBounds;

foreach (IGraphicalComponent graphicalComp in internalBoard.GetGraphicalComponents())

{

possCompBounds = graphicalComp.GetOffsetGraphicalElementBounds();

if (possCompBounds.HasValue)

{

compBounds = possCompBounds.Value;

bounds = new float[]

{

Math.Max(bounds[0], compBounds.Left),

Math.Max(bounds[1], compBounds.Bottom),

};

}

}

internalBoard.ExternalSize = new Size(Math.Max(mousePoint.X - shape.X, (int)Math.Ceiling(bounds[0])), Math.Max(mousePoint.Y - shape.Y, (int)Math.Ceiling(bounds[1])));

ResetSize();

}

else if (dragState.IsInterfaceLoc())

{

Point relToCorner = new Point(mousePoint.X - shape.X, mousePoint.Y - shape.Y);

Dictionary<Board.InterfaceLocation.SideEnum, int> dists = new Dictionary<Board.InterfaceLocation.SideEnum, int> {

{ Board.InterfaceLocation.SideEnum.Left, relToCorner.X },

{ Board.InterfaceLocation.SideEnum.Top, relToCorner.Y },

{ Board.InterfaceLocation.SideEnum.Right, shape.Width - relToCorner.X },

{ Board.InterfaceLocation.SideEnum.Bottom, shape.Height - relToCorner.Y }

};

Board.InterfaceLocation.SideEnum closestSide = dists.Aggregate((kvp1, kvp2) => kvp1.Value < kvp2.Value ? kvp1 : kvp2).Key;

int closestDist = Math.Max(1, closestSide.IsLeftRight() ? Math.Min(relToCorner.Y, shape.Height - 1) : Math.Min(relToCorner.X, shape.Width - 1));

Board.InterfaceLocation.SideEnum actualSide = closestSide;

int actualDist = closestDist;

int distProg = 0, sideProg = 0;

bool lastOutside = false, thisOutside;

while (lastOutside ||

!(GetInterfaceComponent(new Board.InterfaceLocation(actualSide, actualDist)) == null ||

GetInterfaceComponent(new Board.InterfaceLocation(actualSide, actualDist)).GetComponentName() == dragState.GetInterfaceLocName()))

{

actualDist += distProg \* ((2 \* (distProg % 2)) - 1);

distProg++;

thisOutside = actualDist <= 0 || actualDist >= (actualSide.IsLeftRight() ? shape.Height : shape.Width);

if (thisOutside && lastOutside)

{

if (sideProg == 0 || sideProg == 2)

{

actualSide = actualSide.ToggleLeftRight();

} else if (sideProg == 1)

{

actualSide = actualSide.ToggleBottomRight();

} else if (sideProg == 3)

{

actualSide = closestSide; // logically would never happen, as the empty slot it was originally in should still exist. default so edge cases don't crash

}

if (actualSide.IsLeftRight() == closestSide.IsLeftRight())

{

actualDist = closestDist;

} else if (closestSide.IsBottomRight())

{

actualDist = (actualSide.IsLeftRight() ? shape.Height : shape.Width) - 1;

} else

{

actualDist = 1;

}

distProg = 0;

sideProg++;

}

lastOutside = thisOutside;

}

GetInterfaceComponent(dragState.GetInterfaceLocName()).SetInterfaceLocation(new Board.InterfaceLocation(actualSide, actualDist));

} else if (dragState.IsGraphicalComp())

{

IGraphicalComponent graphicalComp = dragState.GetGraphicalComp();

if (mousePoint.X > shape.X && mousePoint.X < shape.X + shape.Width && mousePoint.Y > shape.Y && mousePoint.Y < shape.Y + shape.Height)

{

graphicalComp.SetGraphicalElementLocation(mousePoint);

RectangleF graphicalCompBounds = graphicalComp.GetOffsetGraphicalElementBounds().Value;

foreach (IGraphicalComponent otherGraphicalComp in GetGraphicalComponents().Where(comp => comp.HasGraphics() && comp.GetGraphicalElementLocation().HasValue && comp != graphicalComp))

{

if (otherGraphicalComp.GetOffsetGraphicalElementBounds().Value.IntersectsWith(graphicalCompBounds)) {

graphicalComp.SetGraphicalElementLocation(null);

break;

}

}

} else

{

graphicalComp.SetGraphicalElementLocation(null);

}

ResetSize();

}

Invalidate();

}

}

protected override void OnMouseWheel(MouseEventArgs e)

{

base.OnMouseWheel(e);

Point farCorner = new Point(compDisplayBounds.Width - scale, compDisplayBounds.Height - scale);

if (e.Location.X > compDisplayBounds.Width)

{

graphicalsTransformationMatrix.Translate(0, Math.Sign(e.Delta));

} else

{

if (!(e.Location.X < scale || e.Location.X > farCorner.X || e.Location.Y < scale || e.Location.Y > farCorner.Y))

{

RectangleF? bounds;

foreach (IGraphicalComponent graphicalComp in GetGraphicalComponents())

{

bounds = graphicalComp.GetOffsetScaledGraphicalElementBounds();

if (bounds.HasValue && bounds.Value.Contains(CompDetransformPointF(e.Location)))

{

graphicalComp.SetGraphicalElementScale(graphicalComp.GetGraphicalElementScale() \* (1 + (0.1F \* Math.Sign(e.Delta))));

break;

}

}

}

}

Invalidate();

}

}

##### DragState

private class DragState

{

private bool isInterfaceLoc;

private bool isGraphicalComp;

private bool isSize;

private string interfaceLocName;

private IGraphicalComponent graphicalComp;

private Board.InterfaceLocation interfaceLocReset;

private PointF? graphicalCompReset;

private Size sizeReset;

public DragState()

{

Reset();

}

public bool IsAnything() { return isInterfaceLoc || isGraphicalComp || isSize; }

public bool IsInterfaceLoc() { return isInterfaceLoc; }

public bool IsGraphicalComp() { return isGraphicalComp; }

public bool IsSize() { return isSize; }

public string GetInterfaceLocName() { return interfaceLocName; }

public IGraphicalComponent GetGraphicalComp() { return graphicalComp; }

public Board.InterfaceLocation GetInterfaceLocReset() { return interfaceLocReset; }

public PointF? GetGraphicalCompReset() { return graphicalCompReset; }

public Size GetSizeReset() { return sizeReset; }

public void SelectInterfaceLoc(string name, Board.InterfaceLocation reset) { Reset(); isInterfaceLoc = true; interfaceLocName = name; interfaceLocReset = reset; }

public void SelectGraphicalComp(IGraphicalComponent comp, PointF? reset) { Reset(); isGraphicalComp = true; graphicalComp = comp; graphicalCompReset = reset; }

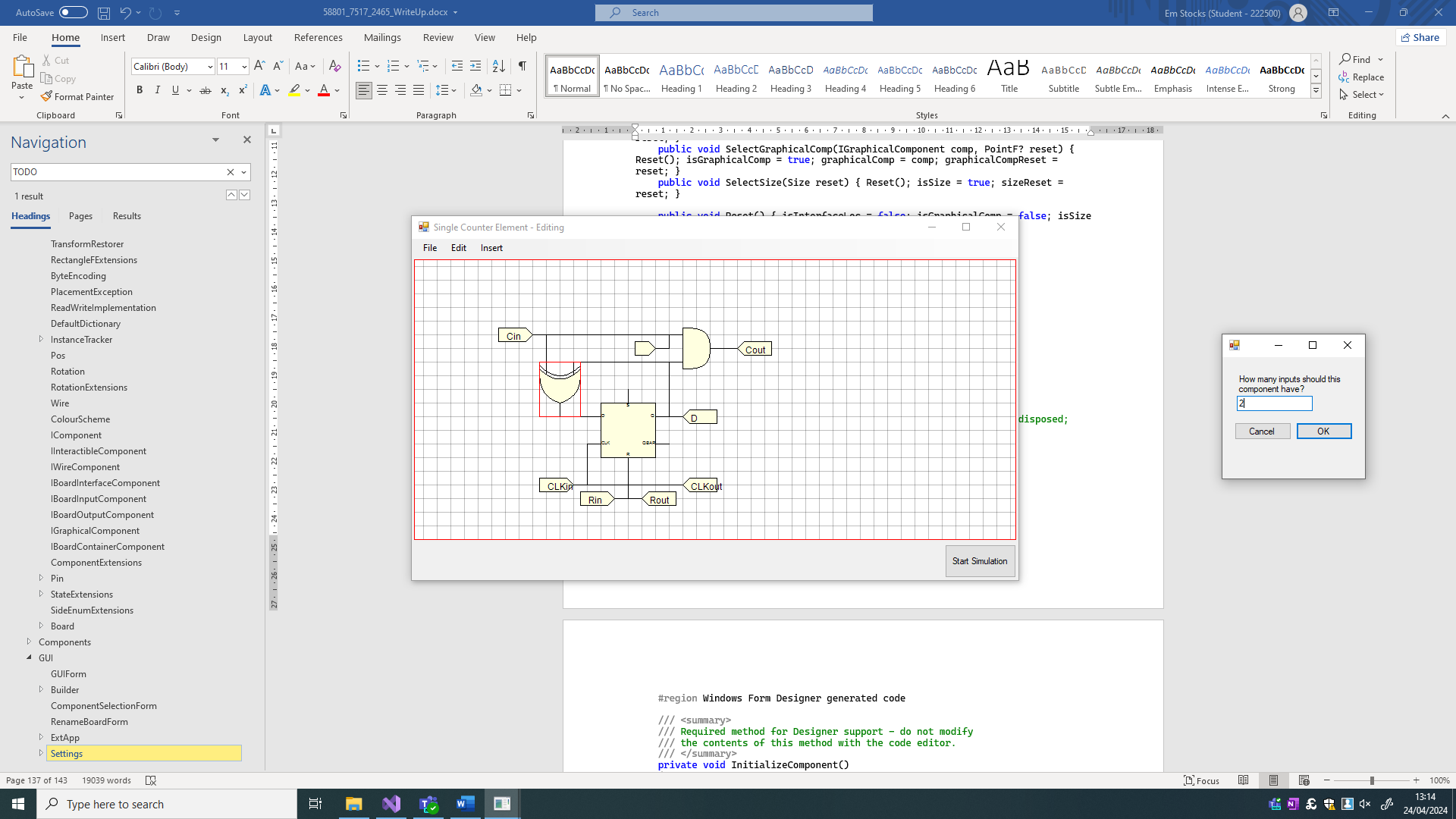
public void SelectSize(Size reset) { Reset(); isSize = true; sizeReset = reset; }

public void Reset() { isInterfaceLoc = false; isGraphicalComp = false; isSize = false; }

}

### Settings

#### SettingsDialog



partial class SettingsDialog

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.flpSettings = new System.Windows.Forms.FlowLayoutPanel();

this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();

this.tblResults = new System.Windows.Forms.TableLayoutPanel();

this.btnOK = new System.Windows.Forms.Button();

this.btnCancel = new System.Windows.Forms.Button();

this.tblMain = new System.Windows.Forms.TableLayoutPanel();

this.flpSettings.SuspendLayout();

this.tblResults.SuspendLayout();

this.tblMain.SuspendLayout();

this.SuspendLayout();

//

// flpSettings

//

this.flpSettings.AutoSize = true;

this.flpSettings.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.flpSettings.Controls.Add(this.tableLayoutPanel1);

this.flpSettings.FlowDirection = System.Windows.Forms.FlowDirection.TopDown;

this.flpSettings.Location = new System.Drawing.Point(3, 3);

this.flpSettings.MinimumSize = new System.Drawing.Size(10, 10);

this.flpSettings.Name = "flpSettings";

this.flpSettings.Size = new System.Drawing.Size(10, 10);

this.flpSettings.TabIndex = 0;

this.flpSettings.WrapContents = false;

//

// tableLayoutPanel1

//

this.tableLayoutPanel1.AutoSize = true;

this.tableLayoutPanel1.ColumnCount = 1;

this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 100F));

this.tableLayoutPanel1.Location = new System.Drawing.Point(3, 3);

this.tableLayoutPanel1.Name = "tableLayoutPanel1";

this.tableLayoutPanel1.RowCount = 2;

this.tableLayoutPanel1.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tableLayoutPanel1.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tableLayoutPanel1.Size = new System.Drawing.Size(0, 0);

this.tableLayoutPanel1.TabIndex = 4;

//

// tblResults

//

this.tblResults.AutoSize = true;

this.tblResults.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.tblResults.ColumnCount = 2;

this.tblResults.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tblResults.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tblResults.Controls.Add(this.btnOK, 1, 0);

this.tblResults.Controls.Add(this.btnCancel, 0, 0);

this.tblResults.Location = new System.Drawing.Point(3, 38);

this.tblResults.Name = "tblResults";

this.tblResults.RowCount = 1;

this.tblResults.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tblResults.Size = new System.Drawing.Size(162, 29);

this.tblResults.TabIndex = 1;

//

// btnOK

//

this.btnOK.DialogResult = System.Windows.Forms.DialogResult.OK;

this.btnOK.Location = new System.Drawing.Point(84, 3);

this.btnOK.Name = "btnOK";

this.btnOK.Size = new System.Drawing.Size(75, 23);

this.btnOK.TabIndex = 0;

this.btnOK.Text = "OK";

this.btnOK.UseVisualStyleBackColor = true;

//

// btnCancel

//

this.btnCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;

this.btnCancel.Location = new System.Drawing.Point(3, 3);

this.btnCancel.Name = "btnCancel";

this.btnCancel.Size = new System.Drawing.Size(75, 23);

this.btnCancel.TabIndex = 1;

this.btnCancel.Text = "Cancel";

this.btnCancel.UseVisualStyleBackColor = true;

//

// tblMain

//

this.tblMain.AutoSize = true;

this.tblMain.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.tblMain.ColumnCount = 1;

this.tblMain.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tblMain.Controls.Add(this.tblResults, 0, 1);

this.tblMain.Controls.Add(this.flpSettings, 0, 0);

this.tblMain.GrowStyle = System.Windows.Forms.TableLayoutPanelGrowStyle.FixedSize;

this.tblMain.Location = new System.Drawing.Point(10, 10);

this.tblMain.Margin = new System.Windows.Forms.Padding(0);

this.tblMain.Name = "tblMain";

this.tblMain.RowCount = 2;

this.tblMain.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tblMain.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));

this.tblMain.Size = new System.Drawing.Size(168, 70);

this.tblMain.TabIndex = 2;

//

// SettingsDialog

//

this.AcceptButton = this.btnOK;

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.AutoSize = true;

this.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;

this.CancelButton = this.btnCancel;

this.ClientSize = new System.Drawing.Size(284, 261);

this.Controls.Add(this.tblMain);

this.KeyPreview = true;

this.Name = "SettingsDialog";

this.Padding = new System.Windows.Forms.Padding(10);

this.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide;

this.flpSettings.ResumeLayout(false);

this.flpSettings.PerformLayout();

this.tblResults.ResumeLayout(false);

this.tblMain.ResumeLayout(false);

this.tblMain.PerformLayout();

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.FlowLayoutPanel flpSettings;

private System.Windows.Forms.TableLayoutPanel tableLayoutPanel1;

private System.Windows.Forms.TableLayoutPanel tblResults;

private System.Windows.Forms.Button btnOK;

private System.Windows.Forms.Button btnCancel;

private System.Windows.Forms.TableLayoutPanel tblMain;

}

public partial class SettingsDialog : Form

{

Label[] labels;

Control[] inputControls;

TableLayoutPanel[] tbls;

public SettingsDialog(string name, ISettingDescription[] settingDescs)

{

InitializeComponent();

Name = name;

labels = new Label[settingDescs.Length];

inputControls = new Control[settingDescs.Length];

tbls = new TableLayoutPanel[settingDescs.Length];

AnchorStyles anchorStyle = AnchorStyles.Bottom | AnchorStyles.Left;

// Code Reference: Dynamic Control generation

for (int i = 0; i < settingDescs.Length; i++) // for every ISettingDescription given in the invocation

{

tbls[i] = new TableLayoutPanel(); // create the containing table

tbls[i].Name = $"tblSetting{i}";

tbls[i].AutoSize = true;

tbls[i].AutoSizeMode = AutoSizeMode.GrowAndShrink;

tbls[i].ColumnCount = 1;

tbls[i].ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 100F));

tbls[i].RowCount = 2;

tbls[i].RowStyles.Add(new RowStyle(SizeType.Percent, 50F));

tbls[i].RowStyles.Add(new RowStyle(SizeType.Percent, 50F));

labels[i] = new Label(); // create the label that will display the prompt for this setting

labels[i].Name = $"lblSetting{i}";

labels[i].Text = settingDescs[i].GetPrompt();

labels[i].AutoSize = true;

labels[i].Anchor = anchorStyle;

inputControls[i] = settingDescs[i].GetInputControl(); // get the ISettingDescription to give a reference to its input control

inputControls[i].Name = $"inpSetting{i}";

tbls[i].Controls.Add(labels[i], 0, 0);

tbls[i].Controls.Add(inputControls[i], 0, 1);

flpSettings.Controls.Add(tbls[i]);

}

}

}

#### ISettingsComponent

interface ISettingsComponent : IComponent

{

ISettingDescription[] GetSettingDescriptions();

void ApplySettings();

}

#### ISettingDescription

public interface ISettingDescription

{

string GetPrompt();

Control GetInputControl();

}

#### SettingsComponentExtensions

static class SettingsComponentExtensions

{

public static void OpenSettings<T>(this T comp) where T : ISettingsComponent

{

SettingsDialog settingsDialog = new SettingsDialog($"{comp.GetComponentID()} Settings", comp.GetSettingDescriptions());

settingsDialog.ShowDialog();

comp.ApplySettings();

}

}

#### SettingDescription

public abstract class SettingDescription<T> : ISettingDescription

{

protected string prompt;

protected T defaultVal;

public string GetPrompt()

{

return prompt;

}

public SettingDescription(string prompt, T defaultValue)

{

this.prompt = prompt;

defaultVal = defaultValue;

}

public abstract Control GetInputControl();

public abstract T GetValue();

}

#### TextBoxSettingDescription

public abstract class TextBoxSettingDescription<T> : SettingDescription<T>

{

protected TextBox inputControl;

public TextBoxSettingDescription(string prompt, T defaultVal) : base(prompt, defaultVal) { }

public override Control GetInputControl()

{

inputControl = new TextBox();

inputControl.Text = defaultVal.ToString();

inputControl.KeyPress += InputControl\_KeyPress;

return inputControl;

}

private void InputControl\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == '\b') { return; }

if (!AllowInput(inputControl.Text, e.KeyChar, inputControl.SelectionStart))

{

e.Handled = true;

}

}

public abstract bool AllowInput(string current, char newChar, int caretIdx);

protected string StripAllBut(string from, string chars)

{

return from.Where(chars.Contains).Select(chr => chr.ToString()).Prepend("").Aggregate((str1, str2) => str1 + str2);

}

}

#### CharLimitTextBoxSettingDescription

public abstract class CharLimitTextBoxSettingDescription<T> : TextBoxSettingDescription<T>

{

protected string CharLimit;

public CharLimitTextBoxSettingDescription(string prompt, T defaultVal, string charLimit) : base(prompt, defaultVal)

{

CharLimit = charLimit;

}

public override bool AllowInput(string current, char newChar, int caretIdx)

{

if (CharLimit.Contains(newChar))

{

return true;

}

return false;

}

}

#### PositiveIntSettingDescription

public class PositiveIntSettingDescription : CharLimitTextBoxSettingDescription<int>

{

public PositiveIntSettingDescription(string prompt, int defaultVal = 0) : base(prompt, defaultVal, "0123456789") { }

public override int GetValue()

{

return int.Parse(inputControl.Text);

}

}

#### SignedIntSettingDescription

public class SignedIntSettingDescription : PositiveIntSettingDescription

{

public SignedIntSettingDescription(string prompt, int defaultVal = 0) : base(prompt, defaultVal) { }

public override bool AllowInput(string current, char newChar, int caretIdx)

{

if (newChar == '-' && caretIdx == 0 && !current.Contains('-'))

{

return true;

}

return base.AllowInput(current, newChar, caretIdx);

}

}

#### NameSettingDescription

public class NameSettingDescription : CharLimitTextBoxSettingDescription<string>

{

public NameSettingDescription(string prompt, string defaultVal) : base(prompt, defaultVal, "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") { }

public override string GetValue()

{

return inputControl.Text;

}

}

#### EnumSettingDescription

public class EnumSettingDescription<E> : SettingDescription<E> where E : Enum {

protected ComboBox inputControl;

public EnumSettingDescription(string prompt, E defaultVal = default) : base(prompt, defaultVal) { }

public override Control GetInputControl()

{

inputControl = new ComboBox();

inputControl.DropDownStyle = ComboBoxStyle.DropDownList;

inputControl.Items.AddRange(Enum.GetValues(typeof(E)).OfType<E>().OfType<object>().ToArray());

inputControl.SelectedItem = defaultVal;

return inputControl;

}

public override E GetValue()

{

return (E)inputControl.SelectedItem;

}

}