```
using System.Drawing;
using System.Windows.Forms;
using CASP_Standalone_Implementation.Src;
using Newtonsoft.Json.Linq;
using System.Collections.Generic;
using System;
using System.Linq;
using System.Drawing.Drawing2D;

namespace CASP_Standalone_Implementation.Forms
{
    public partial class CASP_OutlineForm : CASP_OutputForm // Form
    {
        Pen blockPen = Pens.Gray;

        public CASP_OutlineForm()
        {
            InitializeComponent();
        }

        public override void Set_CASP_Output(JObject CASP_Response)
        {
            List<OutlineGraph> graphs = ParseResponse(CASP_Response);
            int x = 0;

            for (int i = 0; i < graphs.Count; i++)
            {
                Panel p = BreadthFirstDraw(graphs[i]);
                //p.BorderStyle = BorderStyle.FixedSingle;
                p.Location = new Point(x, 0);
                FlowPanel.Controls.Add(p);

                x += p.Width + 20;
                graphs[i].Reset();
            }

        }

        private FlowBlock DrawNode(OutlineNode node, FlowBlock parent, Panel panel, int minX, int y, out Point newPoint)
        {

            node.drawn = true;
            FlowBlock block = GetFlowBlock(node);
            panel.Controls.Add(block);
            int preferredX;

            if (parent != null)
            {
                parent.children.Add(block);
                block.parent = parent;
                preferredX = parent.Center.X - block.Width / 2;
            }
            else
            {
                preferredX = minX;
            }

            block.Location = new Point(Math.Max(minX, preferredX), y);

            newPoint = new Point(block.Right, block.Bottom);
            return block;
        }

        private class node
        {
            public FlowBlock parentFlow;
```

```
67              public List<OutlineNode> children;
68          }
69
70          private Panel BreadthFirstDraw(OutlineGraph graph)
71          {
72              OutlineNode head = graph.nodes[0];
73
74              Panel panel = new Panel();
75              panel.AutoSize = true;
76              int yBuff = 30;
77              int xBuff = 30;
78
79              int y = yBuff;
80              List<FlowBlock> blocks = new List<FlowBlock>();
81              Dictionary<int, FlowBlock> blockDictionary = new Dictionary<int, FlowBlock>();
82              List<List<node>> levels = new List<List<node>>() { new List<node>() { new node() { parentFlow = null, children = new List<OutlineNode>() { head } } } };
83              for (int i = 0; i < levels.Count; i++)
84              {
85
86                  List<node> nodes = levels[i];
87                  int levelY = y;
88                  int minX = xBuff;
89                  for (int k = 0; k < nodes.Count; k++)
90                  {
91                      List<OutlineNode> n = nodes[k].children;
92                      FlowBlock parent = nodes[k].parentFlow;
93
94                      for (int j = 0; j < n.Count; j++)
95                      {
96                          OutlineNode node = n[j];
97                          Point newCoords;
98
99                          FlowBlock block = DrawNode(node, parent, panel, minX, levelY, out newCoords);
100
101                         // TODO need to work on decisions
102                         if (blockDictionary.ContainsKey(node.index))
103                         {
104                             FlowBlock old = blockDictionary[node.index];
105                             blockDictionary.Remove(node.index);
106                             blocks.Remove(old);
107                             panel.Controls.Remove(old);
108                         }
109                         else
110                         {
111                             node newNode = new node
112                             {
113                                 parentFlow = block,
114                                 children = node.edges
115                                 .Where(e => !e.target.drawn)
116                                 .Select(e => e.target)
117                                 .ToList()
118                             };
119
120                             if (levels.Count > i + 1)
121                                 levels[i + 1].Add(newNode);
122                             else
123                                 levels.Add(new List<node>() { newNode });
124                         }
125
126                         blockDictionary.Add(node.index, block);
127                         blocks.Add(block);
128
129                         minX = block.Right + xBuff;
130                         if (newCoords.Y > y)
131                             y = newCoords.Y;
132
133                     }
```

```
134                 }
135 
136                 y += yBuff;
137             }
138 
139             for (int i = 0; i < graph.edges.Count; i++)
140             {
141                 OutlineEdge edge = graph.edges[i];
142                 FlowBlock source = blockDictionary[edge.source.index];
143                 FlowBlock target = blockDictionary[edge.target.index];
144 
145                 if (!source.ConnectTo(target, edge.text))
146                 {
147                     // uh-oh... not enough space on the node. Should only happen on switch, which we don't have
148                 }
149             }
150 
151             RenderEdges(blocks, panel);
152 
153             return panel;
154         }
155 
156         private void RenderEdges(List<FlowBlock> blocks, Panel panel)
157         {
158             panel.Paint += (object sender, PaintEventArgs e) =>
159             {
160                 for (int i = 0; i < blocks.Count; i++)
161                     blocks[i].RenderEdgeGraphics(e.Graphics);
162             };
163         }
164 
165         FlowBlock GetFlowBlock(OutlineNode node)
166         {
167             FlowBlock block = null;
168             switch (node.type)
169             {
170                 case BlockType.Decision:
171                     block = GetFlowDecision(node.text);
172                     break;
173                 case BlockType.End:
174                     block = GetFlowEnd(node.text);
175                     break;
176                 case BlockType.EndDecision:
177                     block = GetFlowSink(node.text);
178                     break;
179                 //case BlockType.IO:
180                 //    block = GetFlowDecision(node.text);
181                 //    break;
182                 case BlockType.Loop:
183                     block = GetFlowLoop(node.text);
184                     break;
185                 case BlockType.MethodCall:
186                     block = GetFlowMethod(node.text);
187                     break;
188                 case BlockType.Process:
189                     block = GetFlowProcess(node.text);
190                     break;
191                 case BlockType.Start:
192                     block = GetFlowEnd(node.text);
193                     break;
194                 default:
195                     block = new FlowBlock();
196                     break;
197             }
198             block.UpdateSockets();
199             block.id = node.index;
200             block.type = node.type;
```

```
201
202             return block;
203         }
204
205         private List<OutlineGraph> ParseResponse(JObject CASP_Response)
206         {
207             JObject data = (JObject)CASP_Response["Data"];
208             JArray outlines = (JArray)data["Outlines"];
209
210             List<OutlineGraph> graphs = new List<OutlineGraph>();
211
212             if (outlines != null)
213             {
214
215                 for (int i = 0; i < outlines.Count; i++)
216                 {
217                     List<dynamic> edgeList = new List<dynamic>();
218                     JArray o = (JArray)outlines[i];
219
220                     OutlineGraph graph = new OutlineGraph();
221
222                     for (int j = 0; j < o.Count; j++)
223                     {
224                         JObject node = (JObject)o[j];
225                         string nodeText = (string)node["data"];
226                         BlockType nodeType = (BlockType)Enum.Parse(typeof(BlockType), (string)node["type"]);
227                         JArray edges = (JArray)node["edges"];
228
229                         graph.AddNode(new OutlineNode() { text = nodeText, type = nodeType });
230
231                         for (int k = 0; k < edges.Count; k++)
232                         {
233                             JObject edge = (JObject)edges[k];
234                             int source = (int)edge["source"];
235                             int target = (int)edge["target"];
236                             string edgeText = (string)edge["data"];
237
238                             edgeList.Add(new { source = source, target = target, text = edgeText });
239                         }
240                     }
241
242                     for (int j = 0; j < edgeList.Count; j++)
243                     {
244                         graph.AddEdge(edgeList[j].source, edgeList[j].target, edgeList[j].text);
245                     }
246
247                     graphs.Add(graph);
248                 }
249
250                 if (graphs.Count == 0)
251                 {
252                     OutlineGraph graph = new OutlineGraph();
253                     graph.AddNode(new OutlineNode()
254                     {
255                         index = 0,
256                         text = "No flowchart data\nto display!",
257                         type = BlockType.Process
258                     });
259                     graphs.Add(graph);
260                 }
261             }
262
263             return graphs;
264         }
265
266         FlowBlock GetFlowEnd(string text)
267         {
```

```
268             FlowBlock flowblock = CreateFlowblock(text);
269             flowblock.Paint += PaintFlowblockEnd;
270             return flowblock;
271         }
272 
273         FlowBlock GetFlowProcess(string text)
274         {
275             FlowBlock flowblock = CreateFlowblock(text);
276             flowblock.Paint += PaintFlowblockProcess;
277             return flowblock;
278         }
279 
280         FlowBlock GetFlowDecision(string text)
281         {
282             FlowBlock flowblock = CreateFlowblock(text);
283             flowblock.Paint += PaintFlowblockDecision;
284             return flowblock;
285         }
286 
287         FlowBlock GetFlowSink(string text)
288         {
289             FlowBlock flowblock = CreateFlowblock("");
290             flowblock.Width = flowblock.Height = 53;
291             flowblock.Paint += PaintFlowblockSink;
292             return flowblock;
293         }
294 
295         FlowBlock GetFlowMethod(string text)
296         {
297             FlowBlock flowblock = CreateFlowblock(text);
298             flowblock.Paint += PaintFlowblockMethod;
299             return flowblock;
300         }
301 
302         FlowBlock GetFlowLoop(string text)
303         {
304             FlowBlock flowblock = CreateFlowblock(text);
305             flowblock.Paint += PaintFlowblockLoop;
306             return flowblock;
307         }
308 
309         private void PaintFlowblockEnd(object sender, PaintEventArgs e)
310         {
311             int left, right, top, bottom, centerX, centerY;
312             FlowBlock flowblock = ReadFlowblockData(sender, out left, out right, out top, out bottom, out centerX, out centerY);
313 
314             Graphics g = e.Graphics;
315             g.DrawArc(blockPen, new Rectangle(left - 5, top, 10, bottom - top), 90, 180);
316             g.DrawLine(blockPen, left, top, right, top);
317             g.DrawArc(blockPen, new Rectangle(right - 5, top, 10, bottom - top), -90, 180);
318             g.DrawLine(blockPen, right, bottom, left, bottom);
319         }
320 
321         private void PaintFlowblockProcess(object sender, PaintEventArgs e)
322         {
323             int left, right, top, bottom, centerX, centerY;
324             FlowBlock flowblock = ReadFlowblockData(sender, out left, out right, out top, out bottom, out centerX, out centerY);
325 
326             Graphics g = e.Graphics;
327             g.DrawLine(blockPen, left, top, right, top);
328             g.DrawLine(blockPen, right, top, right, bottom);
329             g.DrawLine(blockPen, right, bottom, left, bottom);
330             g.DrawLine(blockPen, left, bottom, left, top);
331         }
332 
333         private void PaintFlowblockDecision(object sender, PaintEventArgs e)
334         {
```

```
335             int left, right, top, bottom, centerX, centerY;
336             FlowBlock flowblock = ReadFlowblockData(sender, out left, out right, out top, out bottom, out centerX, out centerY);
337 
338             Graphics g = e.Graphics;
339             g.DrawLine(blockPen, centerX, top - 8, right + 8, centerY);
340             g.DrawLine(blockPen, right + 8, centerY, centerX, bottom + 8);
341             g.DrawLine(blockPen, centerX, bottom + 8, left - 8, centerY);
342             g.DrawLine(blockPen, left - 8, centerY, centerX, top - 8);
343         }
344 
345         private void PaintFlowblockMethod(object sender, PaintEventArgs e)
346         {
347             int left, right, top, bottom, centerX, centerY;
348             FlowBlock flowblock = ReadFlowblockData(sender, out left, out right, out top, out bottom, out centerX, out centerY);
349 
350             Graphics g = e.Graphics;
351             g.DrawLine(blockPen, left - 5, top, right + 5, top);
352             g.DrawLine(blockPen, right + 5, top, right + 5, bottom);
353             g.DrawLine(blockPen, right, top, right, bottom);
354             g.DrawLine(blockPen, right + 5, bottom, left - 5, bottom);
355             g.DrawLine(blockPen, left - 5, bottom, left - 5, top);
356             g.DrawLine(blockPen, left, bottom, left, top);
357         }
358 
359         private void PaintFlowblockLoop(object sender, PaintEventArgs e)
360         {
361             int left, right, top, bottom, centerX, centerY;
362             FlowBlock flowblock = ReadFlowblockData(sender, out left, out right, out top, out bottom, out centerX, out centerY);
363 
364             Graphics g = e.Graphics;
365             g.DrawLine(blockPen, left + 5, top, right - 5, top);
366             g.DrawLine(blockPen, right - 5, top, right + 5, centerY);
367             g.DrawLine(blockPen, right + 5, centerY, right - 5, bottom);
368             g.DrawLine(blockPen, right - 5, bottom, left + 5, bottom);
369             g.DrawLine(blockPen, left + 5, bottom, left - 5, centerY);
370             g.DrawLine(blockPen, left - 5, centerY, left + 5, top);
371         }
372 
373         private void PaintFlowblockSink(object sender, PaintEventArgs e)
374         {
375             int left, right, top, bottom, centerX, centerY;
376             FlowBlock flowblock = ReadFlowblockData(sender, out left, out right, out top, out bottom, out centerX, out centerY);
377 
378             Graphics g = e.Graphics;
379             g.DrawEllipse(blockPen, new Rectangle(left, top, right - left, bottom - top));
380         }
381 
382 
383         private FlowBlock CreateFlowblock(string text)
384         {
385             FlowBlock flowblock = new FlowBlock();
386             flowblock.Text = text;
387             flowblock.AutoSize = true;
388 
389             FlowPanel.Controls.Add(flowblock);
390             int width = flowblock.Width;
391             int height = flowblock.Height;
392             FlowPanel.Controls.Remove(flowblock);
393 
394             flowblock.BackColor = Color.Transparent;
395             flowblock.TextAlign = ContentAlignment.MiddleCenter;
396             flowblock.AutoSize = false;
397             flowblock.Width = width + 40;
398             flowblock.Height = height + 40;
399 
400             flowblock.Cursor = Cursors.Hand;
401 
```

```
402             //flowblock.BorderStyle = BorderStyle.Fixed3D;
403 
404             return flowblock;
405         }
406 
407         private FlowBlock ReadFlowblockData(object sender, out int left, out int right, out int top, out int bottom, out int centerX, out int centerY)
408         {
409             FlowBlock flowblock = sender as FlowBlock;
410 
411             int width = flowblock.Width;
412             int height = flowblock.Height;
413 
414             left = 10;
415             right = width - 10 - 2;
416             top = 10;
417             bottom = height - 10 - 2;
418             centerX = left + (right - left) / 2;
419             centerY = top + (bottom - top) / 2;
420 
421             return flowblock;
422         }
423     }
424 }
```