```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.Drawing;
 4 using System.Drawing.Drawing2D;
 5 using System.Linq;
 6 using System.Text;
 7 using System.Threading.Tasks;
 8 using System.Windows.Forms;
 9
10 namespace CASP_Standalone_Implementation.Src
11 {
12     public enum BlockType { Start, MethodCall, Process, Loop, Decision, EndDecision, IO, End };
13
14     public class OutlineGraph
15     {
16         public List<OutlineNode> nodes = new List<OutlineNode>();
17         public List<OutlineEdge> edges = new List<OutlineEdge>();
18
19         public OutlineNode AddNode(OutlineNode node)
20         {
21             node.index = nodes.Count;
22             nodes.Add(node);
23             return node;
24         }
25
26         public OutlineEdge AddEdge(int sourceIndex, int targetIndex, string text = "")
27         {
28             OutlineEdge edge = nodes[sourceIndex].AddEdge(nodes[targetIndex], text);
29             edges.Add(edge);
30             return edge;
31         }
32
33         public void Reset()
34         {
35             for (int i = 0; i < nodes.Count; i++)
36             {
37                 nodes[i].drawn = false;
38             }
39         }
40     }
41
42     public class OutlineNode
43     {
44         public bool drawn = false;
45         public int index;
46         public string text;
47         public BlockType type;
48         public List<OutlineEdge> edges = new List<OutlineEdge>();
49
50         public OutlineEdge AddEdge(OutlineNode target, string text = "")
51         {
52             OutlineEdge edge = new OutlineEdge() { source = this, target = target, text = text };
53             edges.Add(edge);
54             return edge;
55         }
56     }
57
58     public class OutlineEdge
59     {
60         public string text;
61         public OutlineNode source;
62         public OutlineNode target;
63     }
64
65     public class FlowBlock : Label
66     {
```

```
67          Socket TopSocket;
68          Socket BottomSocket;
69          Socket LeftSocket;
70          Socket RightSocket;
71
72          public bool MouseOver = false;
73          bool renderingSockets = false;
74          public BlockType type;
75          public FlowBlock parent = null;
76          public List<FlowBlock> children = new List<FlowBlock>();
77
78          public List<FlowBlock> siblings
79          {
80              get
81              {
82                  List<FlowBlock> sibs = new List<FlowBlock>();
83                  if (parent != null)
84                  {
85                      for (int i = 0; i < parent.children.Count; i++)
86                      {
87                          if (parent.children[i] != this)
88                              sibs.Add(parent.children[i]);
89                      }
90                  }
91                  return sibs;
92              }
93          }
94
95          public int id;
96
97          public Point Center
98          {
99              get
100             {
101                 return new Point(Width / 2 + Location.X, Height / 2 + Location.Y);
102             }
103         }
104
105
106         public bool RenderSockets
107         {
108             get
109             {
110                 return renderingSockets;
111             }
112             set
113             {
114                 if (value != renderingSockets)
115                 {
116                     if (value)
117                         Paint += FlowBlock_Paint;
118                     else
119                         Paint -= FlowBlock_Paint;
120                 }
121                 renderingSockets = value;
122             }
123         }
124
125         public FlowBlock() : base() {
126             TopSocket = new Socket(this, 90);
127             BottomSocket = new Socket(this, 270);
128             LeftSocket = new Socket(this, 180);
129             RightSocket = new Socket(this, 0);
130             UpdateSockets();
131             //RenderSockets = true;
132         }
133
```

```
134         public void UpdateSockets()
135         {
136             int halfWidth = Width / 2;
137             int halfHeight = Height / 2;
138             int Top = 0;
139             int Bottom = Height;
140             int Left = 0;
141             int Right = Width;
142 
143             TopSocket.SetLocation(halfWidth, Top);
144             BottomSocket.SetLocation(halfWidth, Bottom);
145             LeftSocket.SetLocation(Left, halfHeight);
146             RightSocket.SetLocation(Right, halfHeight);
147         }
148 
149         public Socket ClosestSocketToPoint(Point pt)
150         {
151             Socket closest = TopSocket;
152             double min = TopSocket.DistanceToPoint(pt);
153             double next = BottomSocket.DistanceToPoint(pt);
154             if (next < min)
155             {
156                 min = next;
157                 closest = BottomSocket;
158             }
159             next = LeftSocket.DistanceToPoint(pt);
160             if (next < min)
161             {
162                 min = next;
163                 closest = LeftSocket;
164             }
165             next = RightSocket.DistanceToPoint(pt);
166             if (next < min)
167             {
168                 min = next;
169                 closest = RightSocket;
170             }
171             return closest;
172         }
173 
174         public Socket ClosestUnusedSocketToPoint(Point pt)
175         {
176             Socket closest = null;
177             double min = double.PositiveInfinity;
178             double next = 0;
179 
180             if (TopSocket.Unused)
181             {
182                 next = TopSocket.DistanceToPoint(pt);
183                 if (next < min)
184                 {
185                     min = next;
186                     closest = TopSocket;
187                 }
188             }
189             if (BottomSocket.Unused)
190             {
191                 next = BottomSocket.DistanceToPoint(pt);
192                 if (next < min)
193                 {
194                     min = next;
195                     closest = BottomSocket;
196                 }
197             }
198             if (LeftSocket.Unused) {
199                 next = LeftSocket.DistanceToPoint(pt);
200                 if (next < min)
```

```
201                 {
202                     min = next;
203                     closest = LeftSocket;
204                 }
205             }
206             if (RightSocket.Unused) {
207                 next = RightSocket.DistanceToPoint(pt);
208                 if (next < min)
209                 {
210                     min = next;
211                     closest = RightSocket;
212                 }
213             }
214             return closest;
215         }
216 
217         void ClosestSocketPair(FlowBlock block, out Socket sourceSocket, out Socket targetSocket)
218         {
219             Socket source = null;
220             Socket target = null;
221 
222             double min = double.PositiveInfinity;
223             double next;
224 
225             if (TopSocket.Unused)
226             {
227                 Socket temp = block.type != BlockType.EndDecision ?
228                     block.ClosestUnusedSocketToPoint(TopSocket.Location) :
229                     block.ClosestSocketToPoint(TopSocket.Location);
230                 if (temp != null)
231                 {
232                     next = TopSocket.DistanceToPoint(temp.Location);
233                     if (next < min)
234                     {
235                         min = next;
236                         source = TopSocket;
237                         target = temp;
238                     }
239                 }
240             }
241 
242             if (BottomSocket.Unused)
243             {
244                 Socket temp = block.type != BlockType.EndDecision ?
245                     block.ClosestUnusedSocketToPoint(BottomSocket.Location) :
246                     block.ClosestSocketToPoint(BottomSocket.Location);
247                 if (temp != null)
248                 {
249                     next = BottomSocket.DistanceToPoint(temp.Location);
250                     if (next < min)
251                     {
252                         min = next;
253                         source = BottomSocket;
254                         target = temp;
255                     }
256                 }
257             }
258 
259             if (LeftSocket.Unused)
260             {
261                 Socket temp = block.type != BlockType.EndDecision ?
262                     block.ClosestUnusedSocketToPoint(LeftSocket.Location) :
263                     block.ClosestSocketToPoint(LeftSocket.Location);
264                 if (temp != null)
265                 {
266                     next = LeftSocket.DistanceToPoint(temp.Location);
267                     if (next < min)
```

```
268                     {
269                         min = next;
270                         source = LeftSocket;
271                         target = temp;
272                     }
273                 }
274             }
275 
276             if (RightSocket.Unused)
277             {
278                 Socket temp = block.type != BlockType.EndDecision ?
279                     block.ClosestUnusedSocketToPoint(RightSocket.Location) :
280                     block.ClosestSocketToPoint(RightSocket.Location);
281                 if (temp != null)
282                 {
283                     next = RightSocket.DistanceToPoint(temp.Location);
284                     if (next < min)
285                     {
286                         min = next;
287                         source = RightSocket;
288                         target = temp;
289                     }
290                 }
291             }
292 
293             sourceSocket = source;
294             targetSocket = target;
295         }
296 
297         public bool ConnectTo(FlowBlock block, string data = "")
298         {
299             Socket sourceSocket;
300             Socket targetSocket;
301             ClosestSocketPair(block, out sourceSocket, out targetSocket);
302 
303             if (sourceSocket != null && targetSocket != null)
304             {
305                 Connector c = new Connector()
306                 {
307                     data = data
308                 };
309                 sourceSocket.ConnectAsSource(c);
310                 targetSocket.ConnectAsTarget(c);
311                 return true;
312             }
313             return false;
314         }
315 
316         private void FlowBlock_Paint(object sender, PaintEventArgs e)
317         {
318             Pen pen = Pens.Black;
319             Brush brush = Brushes.Black;
320 
321             int size = 10;
322             int hs = size / 2;
323 
324             if (TopSocket.Unused)
325                 e.Graphics.DrawEllipse(pen, new Rectangle(TopSocket.LocalLocation.X - hs, TopSocket.LocalLocation.Y - hs, size, size));
326             else
327                 e.Graphics.FillEllipse(brush, new Rectangle(TopSocket.LocalLocation.X - hs, TopSocket.LocalLocation.Y - hs, size, size));
328 
329             if (BottomSocket.Unused)
330                 e.Graphics.DrawEllipse(pen, new Rectangle(BottomSocket.LocalLocation.X - hs, BottomSocket.LocalLocation.Y - hs, size, size));
331             else
332                 e.Graphics.FillEllipse(brush, new Rectangle(BottomSocket.LocalLocation.X - hs, BottomSocket.LocalLocation.Y - hs, size, size));
333 
334             if (LeftSocket.Unused)
```

```
335                 e.Graphics.DrawEllipse(pen, new Rectangle(LeftSocket.LocalLocation.X - hs, LeftSocket.LocalLocation.Y - hs, size, size));
336             else
337                 e.Graphics.FillEllipse(brush, new Rectangle(LeftSocket.LocalLocation.X - hs, LeftSocket.LocalLocation.Y - hs, size, size));
338 
339             if (RightSocket.Unused)
340                 e.Graphics.DrawEllipse(pen, new Rectangle(RightSocket.LocalLocation.X - hs, RightSocket.LocalLocation.Y - hs, size, size));
341             else
342                 e.Graphics.FillEllipse(brush, new Rectangle(RightSocket.LocalLocation.X - hs, RightSocket.LocalLocation.Y - hs, size, size));
343 
344         }
345 
346         public void RenderEdgeGraphics(Graphics g)
347         {
348             if (!TopSocket.Unused && TopSocket.isSource)
349                 TopSocket.Connector.RenderGraphicsPath(g);
350 
351             if (!BottomSocket.Unused && BottomSocket.isSource)
352                 BottomSocket.Connector.RenderGraphicsPath(g);
353 
354             if (!LeftSocket.Unused && LeftSocket.isSource)
355                 LeftSocket.Connector.RenderGraphicsPath(g);
356 
357             if (!RightSocket.Unused && RightSocket.isSource)
358                 RightSocket.Connector.RenderGraphicsPath(g);
359         }
360     }
361 
362     public class Socket
363     {
364         public FlowBlock FlowBlock;
365         public Point LocalLocation;
366         public Connector Connector = null;
367         public double angle = 0;
368         public double outAngle
369         {
370             get
371             {
372                 return angle % 360;
373             }
374         }
375         public double inAngle
376         {
377             get
378             {
379                 return (outAngle + 180) % 360;
380             }
381         }
382 
383         public bool isSource
384         {
385             get
386             {
387                 if (Connector != null)
388                     return Connector.Source == this;
389                 return false;
390             }
391         }
392 
393         public bool isTarget
394         {
395             get
396             {
397                 if (Connector != null)
398                     return Connector.Target == this;
399                 return false;
400             }
401         }
```

```
402
403         public Point Location
404         {
405             get
406             {
407                 Point pt = new Point(0, 0);
408                 pt.Offset(FlowBlock.Location);
409                 pt.Offset(LocalLocation);
410                 return pt;
411             }
412         }
413
414         public bool Unused
415         {
416             get
417             {
418                 return Connector == null;
419             }
420         }
421
422         public Socket(FlowBlock parent, double angle, int localX = 0, int localY = 0)
423         {
424             FlowBlock = parent;
425             LocalLocation = new Point(localX, localY);
426             this.angle = angle;
427         }
428
429         public void SetLocation(int localX, int localY)
430         {
431             LocalLocation = new Point(localX, localY);
432         }
433
434         public double DistanceToPoint(Point pt)
435         {
436             return Math.Sqrt(Math.Pow(pt.X - Location.X, 2) + Math.Pow(pt.Y - Location.Y, 2));
437         }
438
439         public bool ConnectAsSource(Connector connector)
440         {
441             //if (Connector == null)
442             //{
443             //    if (connector.Source != null)
444             //        connector.Source.Connector = null;
445                 connector.Source = this;
446                 Connector = connector;
447                 return true;
448             //}
449             //return false;
450         }
451
452         public bool ConnectAsTarget(Connector connector)
453         {
454             //if (Connector == null)
455             //{
456             //    if (connector.Target != null)
457             //        connector.Target.Connector = null;
458                 connector.Target = this;
459                 Connector = connector;
460                 return true;
461             //}
462             //return false;
463         }
464
465     }
466
467     public class Connector
468     {
```

```
469         public Socket Source;
470         public Socket Target;
471 
472         public string data;
473 
474         public void RenderGraphicsPath(Graphics g)
475         {
476             if (Source != null && Target != null)
477             {
478                 RenderArrow(g, Pens.SlateGray, Brushes.SlateGray, 60, 8);
479 
480                 int fontSize = 10;
481                 PointF pt = Source.Location;
482                 StringFormatFlags flags = StringFormatFlags.NoWrap;
483                 if (Source.outAngle == 0 || Source.outAngle == 180)
484                 {
485                     flags = flags | StringFormatFlags.DirectionVertical;
486                 }
487                 StringFormat sf = new StringFormat(flags);
488                 FontFamily fam = new FontFamily("microsoft sans serif");
489                 Font font = new Font(fam, fontSize);
490 
491                 SizeF size = g.MeasureString(data, font);
492 
493                 if (Source.outAngle == 0)
494                     pt = new PointF(pt.X + 3 - size.Height / 2, pt.Y - size.Width / 2);
495                 else if (Source.outAngle == 180)
496                     pt = new PointF(pt.X - 3 - size.Height / 2, pt.Y - size.Width / 2);
497                 else if (Source.outAngle == 90)
498                     pt = new PointF(pt.X - size.Width / 2, pt.Y - 3 - size.Height / 2);
499                 else if (Source.outAngle == 270)
500                     pt = new PointF(pt.X - size.Width / 2, pt.Y + 3 - size.Height / 2);
501 
502 
503                 g.DrawString(data, font, Brushes.Black, pt, sf);
504             }
505         }
506 
507         private void RenderArrow(Graphics g, Pen pen, Brush brush, int arrowAngle, int arrowLength)
508         {
509             Point end = Source.Location;
510 
511             double outAngle = Source.outAngle;
512             double inAngle = Target.inAngle;
513 
514             // complimentary sides (t & b, l & r)
515             if (outAngle == inAngle)
516             {
517                 // vertical
518                 if (Math.Abs(outAngle % 180) == 90)
519                 {
520                     if (Source.Location.X != Target.Location.X)
521                     {
522                         int hy = Source.Location.Y + (Target.Location.Y - Source.Location.Y) / 2;
523                         Point p1 = Source.Location;
524                         Point p2 = new Point(Source.Location.X, hy);
525                         Point p3 = new Point(Target.Location.X, hy);
526                         g.DrawLine(pen, p1, p2);
527                         g.DrawLine(pen, p2, p3);
528                         end = p3;
529                     }
530                 }
531                 // horizontal
532                 else if (Math.Abs(outAngle % 180) == 0)
533                 {
534                     if (Source.Location.Y != Target.Location.Y)
535                     {
```

```
536                     int hx = Source.Location.X + (Target.Location.X - Source.Location.X) / 2;
537                     Point p1 = Source.Location;
538                     Point p2 = new Point(hx, Source.Location.Y);
539                     Point p3 = new Point(hx, Target.Location.Y);
540                     g.DrawLine(pen, p1, p2);
541                     g.DrawLine(pen, p2, p3);
542                     end = p3;
543                 }
544             }
545         }
546         // opposite sides (l & l, t & t, etc.)
547         else if ((inAngle + 180) % 360 == outAngle)
548         {
549             bool set = false;
550             Point p1 = Source.Location, p2 = new Point(), p3 = new Point();
551             // right
552             if (outAngle == 0)
553             {
554                 set = true;
555                 int maxX = Math.Max(Source.Location.X, Target.Location.X) + 10;
556                 p2 = new Point(maxX, Source.Location.Y);
557                 p3 = new Point(maxX, Target.Location.Y);
558             }
559             // left
560             else if (outAngle == 180)
561             {
562                 set = true;
563                 int minX = Math.Min(Source.Location.X, Target.Location.X) - 10;
564                 p2 = new Point(minX, Source.Location.Y);
565                 p3 = new Point(minX, Target.Location.Y);
566             }
567             // top
568             if (outAngle == 90)
569             {
570                 set = true;
571                 int minY = Math.Max(Source.Location.Y, Target.Location.Y) - 10;
572                 p2 = new Point(Source.Location.X, minY);
573                 p3 = new Point(Target.Location.X, minY);
574             }
575             // bottom
576             else if (outAngle == 270)
577             {
578                 set = true;
579                 int maxY = Math.Min(Source.Location.Y, Target.Location.Y) + 10;
580                 p2 = new Point(Source.Location.X, maxY);
581                 p3 = new Point(Target.Location.X, maxY);
582             }
583 
584             if (set)
585             {
586                 g.DrawLine(pen, p1, p2);
587                 g.DrawLine(pen, p2, p3);
588                 end = p3;
589             }
590         }
591         // different axes
592         else
593         {
594             Point p1 = Source.Location, p2 = new Point(), p3 = new Point(), p4 = new Point();
595             // out right or left
596             if (outAngle == 0 || outAngle == 180)
597             {
598                 // in bottom or top
599                 if (inAngle == 90 || inAngle == 270)
600                 {
601                     bool comp;
602                     int xOff = 10;
```

```
603                     if (outAngle == 0)
604                     {
605                         comp = Source.Location.X < Target.Location.X;
606                     }
607                     else
608                     {
609                         comp = Source.Location.X > Target.Location.X;
610                         xOff *= -1;
611                     }
612 
613                     if (comp)
614                     {
615                         p2 = new Point(Target.Location.X, Source.Location.Y);
616                         g.DrawLine(pen, p1, p2);
617                         end = p2;
618                     }
619                     else
620                     {
621                         int yOff = -10;
622                         if (inAngle == 90)
623                             yOff *= -1;
624 
625                         p2 = new Point(Source.Location.X + xOff, p1.Y);
626                         p3 = new Point(p2.X, Target.Location.Y + yOff);
627                         p4 = new Point(Target.Location.X, p3.Y);
628                         g.DrawLine(pen, p1, p2);
629                         g.DrawLine(pen, p2, p3);
630                         g.DrawLine(pen, p3, p4);
631                         end = p4;
632                     }
633                 }
634             }
635             // out bottom or top
636             if (outAngle == 90 || outAngle == 270)
637             {
638                 // in left or right
639                 if (inAngle == 0 || inAngle == 180)
640                 {
641                     bool comp;
642                     int yOff = 10;
643                     if (outAngle == 90)
644                     {
645                         comp = Source.Location.Y > Target.Location.Y;
646                         yOff *= -1;
647                     }
648                     else
649                     {
650                         comp = Source.Location.Y < Target.Location.Y;
651                     }
652 
653                     if (comp)
654                     {
655                         p2 = new Point(Source.Location.X, Target.Location.Y);
656                         g.DrawLine(pen, p1, p2);
657                         end = p2;
658                     }
659                     else
660                     {
661                         int xOff = 10;
662                         if (inAngle == 90)
663                             xOff *= -1;
664 
665                         p2 = new Point(p1.X, Source.Location.Y + yOff);
666                         p3 = new Point(Target.Location.X + xOff, p2.Y);
667                         p4 = new Point(p3.X, Target.Location.Y);
668                         g.DrawLine(pen, p1, p2);
669                         g.DrawLine(pen, p2, p3);
```

```
670                             g.DrawLine(pen, p3, p4);
671                             end = p4;
672                         }
673                     }
674                 }
675             }
676 
677             Point source = end;
678             Point target = Target.Location;
679 
680             double initialAngleDeg = Math.Atan2(target.Y - source.Y, target.X - source.X) * 180 / Math.PI;
681             double angleLRad = (initialAngleDeg - 180 + arrowAngle / 2) * Math.PI / 180;
682             double angleRRad = (initialAngleDeg - 180 - arrowAngle / 2) * Math.PI / 180;
683 
684             Point arrowL = new Point(target.X + (int)(arrowLength * Math.Cos(angleLRad)), target.Y + (int)(arrowLength * Math.Sin(angleLRad)));
685             Point arrowR = new Point(target.X + (int)(arrowLength * Math.Cos(angleRRad)), target.Y + (int)(arrowLength * Math.Sin(angleRRad)));
686 
687             g.FillPolygon(brush, new Point[] { target, arrowL, arrowR });
688             g.DrawLine(pen, source, target);
689         }
690     }
691 }
```