JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2010**

Vol. 9, No. 2, March-April 2010

EDUCATOR'S CORNER

Darwin's World Simulation in C#: The Model/View Classes

Richard Wiener

1 INTRODUCTION

Please refer to the paper, "Darwin's World Simulation in C#: An Interpreter" from the January/February, 2010 issue of JOT. This paper focuses on the GUI aspects of the implementation.

The two remaining classes are *World* (the model of the Darwin world) and the GUI class *WorldUI*. The *World* classs communicates to the *WorldUI* class, the observer class, by firing events. This model/view separation provides for easier maintenance.

Class *World* is presented below.

```
namespace DarwinsWorld {
    // Model class for WorldUI
    public class World {
        // Fields
        private Random rnd = new Random();
        private List<Creature> list = new List<Creature>();
        private bool stop = false;
        private int index = 0;
        private Thread thread1;

        private event Display display;
        private event DisplayPopulation displayPopulation;

        // Properties
        public List<Creature> List {
            get {
                return list;
            }
        }

        public void InitializeWorld(int numRover,
                                int numAndroid, int numTrap, int numFood,
                             int numHopper, int numChangeDirectionRover) {
            for (int i = 0; i < numRover; i++) {
                int x = rnd.Next(0, 15);
                int y = rnd.Next(0, 15);
                if (Global.creatures[x, y] == null) {
                    Direction direction = RandomDirection();
                    Global.creatures[x, y] =
                                    new Creature("Rover.txt",
                                    new Point(y, x), direction,
                                    Color.Green, CreatureType.Rover);
                    display(false, new Point(0, 0),
                                    new Point(y, x), "R",
                                     Color.Green, direction);
                    list.Add(Global.creatures[x, y]);
                } else {
                    i--;
                }
            }

            for (int i = 0; i < numAndroid; i++) {
                int x = rnd.Next(0, 15);
                int y = rnd.Next(0, 15);
                if (Global.creatures[x, y] == null) {
                    Direction direction = RandomDirection();
                    Global.creatures[x, y] =
                             new Creature("Android.txt",
                             new Point(y, x), direction,
                             Color.Blue, CreatureType.Android);
                    display(false, new Point(0, 0),
                        new Point(y, x), "A",
                        Color.Blue, direction);
                    list.Add(Global.creatures[x, y]);
                } else {
                    i--;
                }
            }

            for (int i = 0; i < numTrap; i++) {
                int x = rnd.Next(0, 15);
                int y = rnd.Next(0, 15);
                if (Global.creatures[x, y] == null) {
                    Direction direction = RandomDirection();
                    Global.creatures[x, y] =
                              new Creature("Trap.txt",
                              new Point(y, x), direction,
                             Color.Red, CreatureType.Trap);
                    display(false, new Point(0, 0),
                            new Point(y, x), "T",
                                                   Color.Red, direction);
                    list.Add(Global.creatures[x, y]);
                } else {
                    i--;
                }
            }

            for (int i = 0; i < numFood; i++) {
```

```
                int x = rnd.Next(0, 15);
                int y = rnd.Next(0, 15);
                if (Global.creatures[x, y] == null) {
                    Direction direction = RandomDirection();
                    Global.creatures[x, y] =
                            new Creature("Food.txt",
                            new Point(y, x), direction,
                            Color.Orange, CreatureType.Food);
                    display(false, new Point(0, 0),
                            new Point(y, x), "F",
                            Color.Orange, direction);
                    list.Add(Global.creatures[x, y]);
                } else {
                    i--;
                }
            }

            for (int i = 0; i < numHopper; i++) {
                int x = rnd.Next(0, 15);
                int y = rnd.Next(0, 15);
                if (Global.creatures[x, y] == null) {
                    Direction direction = RandomDirection();
                    Global.creatures[x, y] =
                       new Creature("Hopper.txt",
                       new Point(y, x),
                       direction, Color.Black, CreatureType.Hopper);
                    display(false, new Point(0, 0),
                                                  new Point(y, x), "H",
                                                  Color.Black, direction);
                    list.Add(Global.creatures[x, y]);
                } else {
                    i--;
                }
            }

            for (int i = 0; i < numChangeDirectionRover; i++) {
                int x = rnd.Next(0, 15);
                int y = rnd.Next(0, 15);
                if (Global.creatures[x, y] == null) {
                    Direction direction = RandomDirection();
                    Global.creatures[x, y] =
                            new Creature("ChangeDirectionRover.txt",
          new Point(y, x), direction,
          Color.Brown, CreatureType.ChangeDirectionRover);
          display(false, new Point(0, 0), new Point(y, x), "C",
                Color.Brown, direction);
          list.Add(Global.creatures[x, y]);
      } else {
          i--;
      }
  }

  thread1 = new Thread(new ThreadStart(RunSimulation));
  thread1.IsBackground = true;

   FillTable();
  }

 public void Register(Display worldDisplay) {
     if (display == null) {
           display += worldDisplay;
      }
 }

 public void Register(DisplayPopulation populationDisplay) {
     if (displayPopulation == null) {
          displayPopulation += populationDisplay;
      }
 }

  public void FireDisplay(bool infect, Point oldPos, Point newPos, String symbol,
                          Color c, Direction d) {
     display(infect, oldPos, newPos, symbol, c, d);
  }

  public void Start() {
     thread1.Start();
  }

  public void RunSimulation() {
       int moveCycle = 0;
       for (; !stop; ) {
            lock (list) {
                Population(moveCycle);
            }
            ShuffleList();
            for (int index = 0;
                  index < list.Count && !stop; index++) {
                lock (list) {
                    list[index].Move(this);
                }
            Thread.Sleep(50);
                    }
                    moveCycle++;
                }
            }

            public void Step() {
                int moveCycle = 0;
                if (index < list.Count) {
                    list[index].Move(this);
                    index++;
                } else {
                    moveCycle++;
                    lock (list) {
                        Population(moveCycle);
                    }
                    ShuffleList();
                    index = 0;
                    list[index].Move(this);
                    index++;
                }
```

```
}

public void Stop() {
    stop = true;
}

private Direction RandomDirection() {
    int value = rnd.Next(4) + 1;
    switch (value) {
        case 1: return Direction.North;
        case 2: return Direction.East;
        case 3: return Direction.South;
        case 4: return Direction.West;
        default: return Direction.North;
    }
}

private void ShuffleList() {
    List<Creature> shuffledList = new List<Creature>();
    while (list.Count > 0) {
        // Choose random index in list
        int index = rnd.Next(0, list.Count);
        shuffledList.Add(list[index]);
        list.RemoveAt(index);
    }
    list = new List<Creature>();
    foreach (Creature creature in shuffledList) {
        list.Add(creature);
    }
}

private void Population(int moveCycle) {
    int android = 0;
    int rover = 0;
    int food = 0;
    int trap = 0;
    int hopper = 0;
    int changeDirectionRover = 0;
    foreach (Creature creature in list) {
        switch (creature.CreatureType) {
            case CreatureType.Android: android++; break;
            case CreatureType.Rover: rover++; break;
            case CreatureType.Food: food++; break;
            case CreatureType.Trap: trap++; break;
            case CreatureType.Hopper: hopper++; break;
             case CreatureType.ChangeDirectionRover:
        changeDirectionRover++;
                                break;
        }
    }
    displayPopulation(moveCycle, android, rover,
                      trap, food, hopper, changeDirectionRover);
}

// Private action commands
private bool Left(Point oldPos, Point newPos,
                    String programFileName) {
    Global.creatures[oldPos.Y, oldPos.X].LeftDirection();
    return false;
}

private bool Right(Point oldPos, Point newPos,
                    String programFileName) {
    Global.creatures[oldPos.Y, oldPos.X].RightDirection();
    return false;
}

private bool Infect(Point oldPos, Point newPos,
         String progamFileName) {
    Global.creatures[newPos.Y, newPos.X].ProgramFileName =
                                        progamFileName;
    Global.creatures[newPos.Y, newPos.X].BuildInstructions();
    return false;
}

private bool IfEmpty(Point oldPos, Point newPos,
                     String programFileName) {
    return Global.creatures[newPos.Y, newPos.X] == null;
}

private bool IfWall(Point oldPos, Point newPos,
        String programFileName) {
    if (Global.creatures[newPos.Y, newPos.X] == null) {
        return false;
    }
    return newPos.X > 14 || newPos.X < 0 ||
                                    newPos.Y < 0 || newPos.Y > 14;
}

private bool IfSame(Point oldPos, Point newPos,
                    String programFileName) {
    if (Global.creatures[newPos.Y, newPos.X] == null) {
        return false;
    }
    return Global.creatures[newPos.Y,newPos.X].
               ProgramFileName.Equals(programFileName);
}

private bool IfEnemy(Point oldPos, Point newPos,
                     String programFileName) {
    if (Global.creatures[newPos.Y, newPos.X] == null) {
        return false;
    } else {
        return
               !Global.creatures[newPos.Y, newPos.X].
               ProgramFileName.Equals(programFileName);
    }
}

private bool Hop(Point oldPos, Point newPos,
                   String programFileName) {
```

```
            Global.creatures[newPos.Y, newPos.X] =
                    Global.creatures[oldPos.Y, oldPos.X];
            Global.creatures[newPos.Y, newPos.X].Position = newPos;
            Global.creatures[oldPos.Y, oldPos.X] = null;
            return false;
        }

        private bool IfRandom(Point oldPos, Point newPos,
                              String programFileName) {
            return rnd.NextDouble() <= 0.5;
        }

        private bool Go(Point oldPos, Point newPos,
                          String programFileName) {
            return true;
        }

        private void FillTable() {
            Global.SetTable(OpCode.Left, Left);
            Global.SetTable(OpCode.Right, Right);
            Global.SetTable(OpCode.Infect, Infect);
            Global.SetTable(OpCode.IfEmpty, IfEmpty);
            Global.SetTable(OpCode.IfWall, IfWall);
            Global.SetTable(OpCode.IfSame, IfSame);
            Global.SetTable(OpCode.IfEnemy, IfEnemy);
            Global.SetTable(OpCode.Hop, Hop);
            Global.SetTable(OpCode.IfRandom, IfRandom);
            Global.SetTable(OpCode.Go, Go);
            Global.SetTable(OpCode.IfEnemyLeft, IfEnemy);
            Global.SetTable(OpCode.IfEnemyRight, IfEnemy);
        }
    }
}
```

The *FillTable* command is executed upon the completion of the *InitializeWorld* method. Each of the concrete action methods that implement the *TakeAction* delegate type are defined in this class.

The *RunSimulation* method takes a census of the creatures contained in the world and fires an event *displayPopulation* received in class *WorldUI*. Then the list is shuffled. Then the *Move* command is invoked on every *Creature* object in the list (held by the *World*).

Finally we present and discuss the GUI class, *WorldUI*. Its details are given below.

```
namespace DarwinsWorld {

    public partial class WorldUI : Form {

        // Fields
        private Graphics panelGraphics;
        private Graphics windowGraphics;
        private World world = new World();
        private Point[] trackAndroid =
                    new Point[4] { new Point(10, 950), new Point(11, 950),
                    new Point(12, 950), new Point(13, 950) };
```

```
private Point[] trackRover = new Point[4] { new Point(10, 950),
               new Point(11, 950), new Point(12, 950),
               new Point(13, 950) };
private Point[] trackTrap = new Point[4] { new Point(10, 950),
               new Point(11, 950), new Point(12, 950),
               new Point(13, 950) };
private Point[] trackFood = new Point[4] { new Point(10, 950),
               new Point(11, 950), new Point(12, 950),
               new Point(13, 950) };
private Point[] trackHopper = new Point[4] {
               new Point(10, 950), new Point(11, 950),
               new Point(12, 950), new Point(13, 950) };
private Point[] trackChangeRover = new Point[4] {
                new Point(10, 950), new Point(11, 950),
                new Point(12, 950), new Point(13, 950) };


 public WorldUI() {
    InitializeComponent();
    panelGraphics = panel.CreateGraphics();
    world.Register(Display);
    world.Register(DisplayPopulationsStats);
    panelGraphics = panel.CreateGraphics();
    windowGraphics = this.CreateGraphics();
}

public void Display(bool infect, Point oldPos,
                Point newPos, String symbol,
               Color c, Direction d) {
    if (!this.InvokeRequired) {
        if (!infect) {
            panelGraphics.FillRectangle
               (new SolidBrush(Color.White),
                new Rectangle(new Point(oldPos.X * 50 + 5,
               oldPos.Y * 50 + 5), new Size(45, 45)));
        } else {
            panelGraphics.FillRectangle(
               new SolidBrush(Color.White),
               new Rectangle(new Point(newPos.X * 50 + 5,
               newPos.Y * 50 + 5), new Size(45, 45)));
        }

        if (d == Direction.North) {
            panelGraphics.FillEllipse(
               new SolidBrush(Color.Red),
               new Rectangle(new Point(newPos.X * 50 + 20,
               newPos.Y * 50 + 5), new Size(5, 5)));
        } else if (d == Direction.East) {
            panelGraphics.FillEllipse(
                new SolidBrush(Color.Red),
                new Rectangle(new Point(newPos.X * 50 + 35,
               newPos.Y * 50 + 20), new Size(5, 5)));
        } else if (d == Direction.South) {
            panelGraphics.FillEllipse(
                new SolidBrush(Color.Red),
                new Rectangle(new Point(newPos.X * 50 + 20,
                newPos.Y * 50 + 35), new Size(5, 5)));
        } else if (d == Direction.West) {
            panelGraphics.FillEllipse(
               new SolidBrush(Color.Red),
               new Rectangle(new Point(newPos.X * 50 + 5,
               newPos.Y * 50 + 20), new Size(5, 5)));
        }
        panelGraphics.DrawString(symbol,
                new Font("Courier", 18),
                new SolidBrush(c), newPos.X * 50 + 10,
                                 newPos.Y * 50 + 10);
    } else {
        Object[] parameters = { infect, oldPos,
                    newPos, symbol, c, d };
        this.Invoke(new DisplayUpdate(Display), parameters);
    }
}


 public void DisplayPopulationsStats(int moveCycle,
        int android, int rover, int trap, int food,
        int hopper, int changeDirectionRover) {
    if (!this.InvokeRequired) {
        moveCycleTextBox.Text = "" + moveCycle;
        androidTextBox.Text = "" + android;
        roverTextBox.Text = "" + rover;
        trapTextBox.Text = "" + trap;
        foodTextBox.Text = "" + food;
        hopperTextBox.Text = "" + hopper;
        changeDirectionRoverTextBox.Text = "" +
               changeDirectionRover;
        UpdateAndroid(moveCycle, android);
        windowGraphics.DrawBezier(
               new Pen(Color.Blue, 1), trackAndroid[0],
               trackAndroid[1], trackAndroid[2], trackAndroid[3]);
        UpdateRover(moveCycle, rover);
        windowGraphics.DrawBezier(new Pen(Color.Green, 1),
               trackRover[0], trackRover[1], trackRover[2],
               trackRover[3]);
        UpdateTrap(moveCycle, trap);
        windowGraphics.DrawBezier(new Pen(Color.Red, 1),
               trackTrap[0], trackTrap[1], trackTrap[2],
               trackTrap[3]);
        UpdateFood(moveCycle, food);
        windowGraphics.DrawBezier(new Pen(Color.Orange, 1),
               trackFood[0], trackFood[1], trackFood[2],
               trackFood[3]);
        UpdateHopper(moveCycle, hopper);
        windowGraphics.DrawBezier(new Pen(Color.Black, 1),
               trackHopper[0], trackHopper[1], trackHopper[2],
               trackHopper[3]);
        UpdateChangeRover(moveCycle, changeDirectionRover);
        windowGraphics.DrawBezier(new Pen(Color.Brown, 1),
               trackChangeRover[0], trackChangeRover[1],
               trackChangeRover[2], trackChangeRover[3]);
    } else {
        Object[] parameters = { moveCycle, android,
               rover, trap, food, hopper, changeDirectionRover };
        this.Invoke(new DisplayPopulationsStatsUpdate(
                DisplayPopulationsStats), parameters);
    }
}

public void DrawGrid() {
    for (int row = 0; row < 15; row++) {
        panelGraphics.DrawLine(
         (new Pen(Color.Black), 0, row * 50, 750, row * 50);
    }
    for (int col = 0; col < 15; col++) {
        panelGraphics.DrawLine(
        new Pen(Color.Black), col * 50, 0, col * 50, 750);
    }
    List<Creature> creatures = world.List;


    foreach (Creature creature in creatures) {
        Display(false, new Point(0, 0), creature.Position,
                creature.ProgramFileName[0].ToString(),
                creature.Color, creature.Direction);
    }
 windowGraphics.DrawLine(new Pen(Color.Black), 10,
                950, 1000, 950);
}

private void initializeGrid_Click(object sender, EventArgs e) {
    initializeGrid.Enabled = false;
    world.InitializeWorld(Convert.ToInt32(roverTextBox.Text),
                          Convert.ToInt32(androidTextBox.Text),
                          Convert.ToInt32(trapTextBox.Text),
                          Convert.ToInt32(foodTextBox.Text),
                          Convert.ToInt32(hopperTextBox.Text),
                          Convert.ToInt32(

        changeDirectionRoverTextBox.Text));
}

private void start_Click(object sender, EventArgs e) {
    start.Enabled = false;
    stepBtn.Enabled = false;
    world.Start();
}

private void stopBtn_Click(object sender, EventArgs e) {
    stopBtn.Enabled = false;
    world.Stop();
}

private void stepBtn_Click(object sender, EventArgs e) {
    start.Enabled = false;
    world.Step();
}

private void panel_Paint(object sender, PaintEventArgs e) {
    DrawGrid();
```

```
        }

        private void UpdateAndroid(int moveCycle, int android) {
            trackAndroid[0] = trackAndroid[1];
            trackAndroid[1] = trackAndroid[2];
            trackAndroid[2] = trackAndroid[3];
            trackAndroid[3] = new Point(10 + moveCycle * 5,
                      950 - android * 5);
        }

        private void UpdateRover(int moveCycle, int rover) {
            trackRover[0] = trackRover[1];
            trackRover[1] = trackRover[2];
            trackRover[2] = trackRover[3];
            trackRover[3] = new Point(10 + moveCycle * 5,
                      950 - rover * 5);
        }

        private void UpdateTrap(int moveCycle, int trap) {
            trackTrap[0] = trackTrap[1];
            trackTrap[1] = trackTrap[2];
            trackTrap[2] = trackTrap[3];
            trackTrap[3] = new Point(10 + moveCycle * 5,
                      950 - trap * 5);
        }

        private void UpdateFood(int moveCycle, int food) {
            trackFood[0] = trackFood[1];
            trackFood[1] = trackFood[2];
            trackFood[2] = trackFood[3];
            trackFood[3] = new Point(10 + moveCycle * 5,
                      950 - food * 5);
        }

        private void UpdateHopper(int moveCycle, int hopper) {
            trackHopper[0] = trackHopper[1];
            trackHopper[1] = trackHopper[2];
            trackHopper[2] = trackHopper[3];
            trackHopper[3] = new Point(10 + moveCycle * 5,
                       950 - hopper * 5);
        }

        private void UpdateChangeRover(int moveCycle,
                      int changeDirectionRover) {
            trackChangeRover[0] = trackChangeRover[1];
            trackChangeRover[1] = trackChangeRover[2];
            trackChangeRover[2] = trackChangeRover[3];
            trackChangeRover[3] = new Point(10 + moveCycle * 5,
                                                                 950 -
                changeDirectionRover * 5);
        }

        private void WorldUI_FormClosed(object sender,
                FormClosedEventArgs e) {
            world.Stop();
        }
        }
    }
```

The two listener methods *Display* and *DisplayPopulationsStats* are registered with the world object using,

```
    world.Register(Display);
    world.Register(DisplayPopulationsStats);
```

in the *WorldUI* constructor.

Both of these listener methods test whether *InvokeRequired* is true. This property of class *Form*, if true, indicates that a GUI control has been activated from an external thread. If allowed, this would most likely cause a CrossThread exception to be generated at runtime. This in fact was observed in an earlier version of the implementation before making the fix shown in each of these methods.

The "if" block contains the code that would normally been directly invoked on the GUI controls from the external thread. The "else" block reinvokes the listener method so that it appears to originate from the GUI thread. The structure of this code is:

```
    if (!this.InvokeRequired) {
       // Code that would normally be invoked on the control
    } else {
       // Use of Invoke to avoid CrossThread exception
    }
```

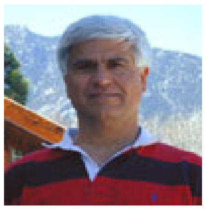
The *Paint* event handler of the panel component is attached to the panel_Paint meth\od.

This *panel_Paint* method is invoked by the runtime system whenever the system detects a need to repaint the panel component. It in turns invokes the *DrawGrid* method. Using the *List* property of the world object, DrawGrid recreates the graphics for each creature held by the *world* object.

So in summary, the *DarwinsWorld* application implements an interpreter that allows users to configure their own creature types. Each line of the user's "program" is parsed to determine the appropriate op code associated with the line of code. Then each op code is used as a key in a standard .NET Dictionary component and associated with an instance of the *TakeAction* delegate. The actual interpretation occurs when the *Move* command, in class *Creature* is invoked. The action associated with the *OpCode* in the line being interpreted is invoked. The result of the action is communicated to the GUI through the *FireDisplay* event invoked on the *world* object passsed as a parameter to the *Move* command.

The important C# programming and software development principles of event handling and model/view separation, use of delegate types as "data" in a dictionary and the construction of a GUI class that listens to events originating in a separate model class thread and using Invoke to protect against a cross-thread exception are all included in this application.

About the author



Richard Wiener is Chair of Computer Science at the University of Colorado at Colorado Springs. He is also the Editor-in-Chief of JOT and former Editor-in-Chief of the Journal of Object Oriented Programming. In addition to University work, Dr. Wiener has authored or co-authored 22 books and works actively as a consultant and software contractor whenever the possibility arises. His latest book, published by Thomson, Course Technology in April 2007, is entitled *Modern Software Development Using C#/.NET*.