Introduction to Game Programming and Robotics

Unit # 8

Connection with Lego Mindstorms

- Step 1: Create partnerships
- Step 2: Add “using” statements
- Step 3: Subscribe to different sensor services
- Step 4: Write code for Event Handlers

Creating Partnerships

- Create partnership with the following services:
  - Light Sensor
  - UltraSonic Sensor
  - NXT Brick
  - Generic Drive

Creating Partnerships (Cont'd)

```
/// NxtLightSensor partner
[Partner("NxtLightSensor", Contract = lightsensor.Contract.Identifier, CreationPolicy =
 PartnerCreationPolicy.UseExisting)]
lightsensor.LightSensorOperations _nxtLightSensorPort = new lightsensor.LightSensorOperations();
lightsensor.LightSensorOperations _nxtLightSensorNotify = new lightsensor.LightSensorOperations();

 /// NxtUltrasonicSensor partner
[Partner("NxtUltrasonicSensor", Contract = sonarsensor.Contract.Identifier, CreationPolicy =
 PartnerCreationPolicy.UseExisting)]
sonarsensor.UltrasonicSensorOperations _nxtUltrasonicSensorPort = new sonarsensor.UltrasonicSensorOperations();
sonarsensor.UltrasonicSensorOperations _nxtUltrasonicSensorNotify = new sonarsensor.UltrasonicSensorOperations();

/// NxtDrive partner
[Partner("Drive", Contract = drive.Contract.Identifier, CreationPolicy = PartnerCreationPolicy.UseExisting)]
drive.DriveOperations _drivePort = new drive.DriveOperations();
drive.DriveOperations _driveNotify = new drive.DriveOperations();

/// Brick partner
[Partner("NxtBrick", Contract = brick.Contract.Identifier, CreationPolicy = PartnerCreationPolicy.UseExisting)]
brick.NxtBrickOperations _nxtBrickPort = new brick.NxtBrickOperations();
brick.NxtBrickOperations _nxtBrickNotify = new brick.NxtBrickOperations();
```

Add "using" Statements

- using submgr = Microsoft.Dss.Services.SubscriptionManager;
- using lightsensor = Microsoft.Robotics.Services.Sample.Lego.Nxt.LightSensor.Proxy;
- using sonarsensor = Microsoft.Robotics.Services.Sample.Lego.Nxt.SonarSensor.Proxy;
- using drive = Microsoft.Robotics.Services.Drive.Proxy;
- using brick = Microsoft.Robotics.Services.Sample.Lego.Nxt.Brick.Proxy;
- using motor = Microsoft.Robotics.Services.Sample.Lego.Nxt.Motor.Proxy;

Subscribe to Services

- _nxtButtonsPort.Subscribe(_nxtButtonsNotify);
- _nxtUltrasonicSensorPort.Subscribe(_nxtUltrasonicSensor Notify);
- _nxtLightSensorPort.Subscribe(_nxtLightSensorNotify);
- _nxtBrickPort.Subscribe(_nxtBrickNotify);
- _drivePort.Subscribe(_driveNotify);

Activate Handlers

```
Activate(Arbiter.Interleave(
            new TeardownReceiverGroup(),
            new ExclusiveReceiverGroup(),
            new ConcurrentReceiverGroup(

    Arbiter.Receive<sonarsensor.SonarSensorUpdate>(true,
        _nxtUltrasonicSensorNotify, UltraSonicHandler),
    Arbiter.Receive<brick.ConnectToHardware>(false,
        _nxtBrickNotify, BrickHandler))
            )
          );
```


BrickHandler()

```
// This code turns the spotlight on
private void BrickHandler(brick.ConnectToHardware
   notification)
      {
         Console.WriteLine("Brick Handler Called ");
         lightsensor.SpotlightRequest spotlightRequest =
                  new lightsensor.SpotlightRequest();
         spotlightRequest.SpotlightOn = true;
         _nxtLightSensorPort.Spotlight(spotlightRequest);
      }
```

Ultrasonic Handler

```
private void UltraSonicHandler(sonarsensor.SonarSensorUpdate notification)
    {

        if (notification.Body.Distance < 50)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Ultrasonic Handler called at " + notification.Body.TimeStamp + " Reading is
                        " + notification.Body.Distance);
            Console.ForegroundColor = ConsoleColor.White;
            _state.driving = true;
            SpawnIterator<double, double>(2, 0.9, DriveDistance);

            while (_state.driving)
            {
            }
            SpawnIterator<double, double>(75, 0.6, MakeTurn);

        }

    }
```


DriveDistance

```
private IEnumerator<ITask> DriveDistance(double distance, double power)
     {
         drive.DriveDistanceRequest distanceRequest = new
    drive.DriveDistanceRequest();
         distanceRequest.Distance = distance;
         distanceRequest.Power = power;

         yield return Arbiter.Choice(_drivePort.DriveDistance(distanceRequest),
            delegate(DefaultUpdateResponseType response) { },
            delegate(Fault fault) { Console.WriteLine ("Unable to drive " + fault); }
           );

     }
```

MakeTurn

```
private IEnumerator<ITask> MakeTurn(double degree, double power)
    {
        drive.RotateDegreesRequest rotateRequest = new
    drive.RotateDegreesRequest();
        rotateRequest.Degrees = degree;
        rotateRequest.Power = power;

        yield return Arbiter.Choice(_drivePort.RotateDegrees(rotateRequest),
           delegate(DefaultUpdateResponseType response) { },
           delegate(Fault fault) { Console.WriteLine("Unable to drive " + fault); }
          );

    }
```


Issues with Running Multiple Instructions in Sequence

- In the previous code, it's not possible to DriveDistance and MakeTurn in sequence.
- The reason is both of them run on separate threads and Arbiter.Choice within the DriveDistance is activated immediately as soon as the drive operation starts.
- The solution is to use DriveDistance and RotateDegrees Handlers as explained on the next slides.

Activate Drive Handlers

- Activate(Arbiter.ReceiveWithIterator<drive.DriveDistance>(true, _driveNotify, DriveDistanceUpdateHandler));
- Activate(Arbiter.ReceiveWithIterator<drive.RotateDegrees>(true, _driveNotify, RotateDegreesUpdateHandler));

DriveDistance Handler

```
private IEnumerator<ITask> DriveDistanceUpdateHandler(drive.DriveDistance distance)
    {

        if (distance.Body.DriveDistanceStage == drive.DriveStage.Canceled)
        {
            Console.WriteLine("Drive Distance Cancelled " + distance.Body.DriveDistanceStage);

        }
        if (distance.Body.DriveDistanceStage == drive.DriveStage.Completed)
        {
            Console.WriteLine("Drive Distance Finished " + distance.Body.DriveDistanceStage + " : " +
DateTime.Now);
            _state.driving = false;
        }

        yield break;

    }
```

RoateDegrees Handler

```
private IEnumerator<ITask> RotateDegreesUpdateHandler(drive.RotateDegrees rotate)
    {
      Console.WriteLine("Rotate Degree Called " + rotate.Body.RotateDegreesStage);
      if (rotate.Body.RotateDegreesStage == drive.DriveStage.Canceled)
      {
        Console.WriteLine("Rotate Degreee Cancelled");
      }
      if (rotate.Body.RotateDegreesStage == drive.DriveStage.Completed)
      {
        Console.WriteLine("Rotate Degrees Finished");
      }
      yield break;

    }
```


Issues with DriveDistance and RotateDegrees

- Till now, both DriveDistance and RotateDegrees work fine in the simulation mode but when run on the actual Lego Mindstorms they don't work as per expected.
- The "DriveInTriangle" Hands On Lab also uses these two handlers.