Software Engineering Method

Assignment2

525910 Hanmoi Choi

Task2

I have used Java Random class's nextGaussian functions to generate random value in accordance to operation profile, which is that a mean 1600C and a standard deviation of 100C.

Implementation is like below

Temperature = 1600 + random.nextGaussian() * deviation

Both 10000 times Iteration.

Build1

	Normal	Degraded	Shutdown
Normal	99.9748%(7936)	0.0126%(1)	0.0126%(1)
Degraded	0.1117%(1)	99.8883%(895)	0 %(0)
Shutdown	0%(0)	0%(0)	100%(1166)

Build2

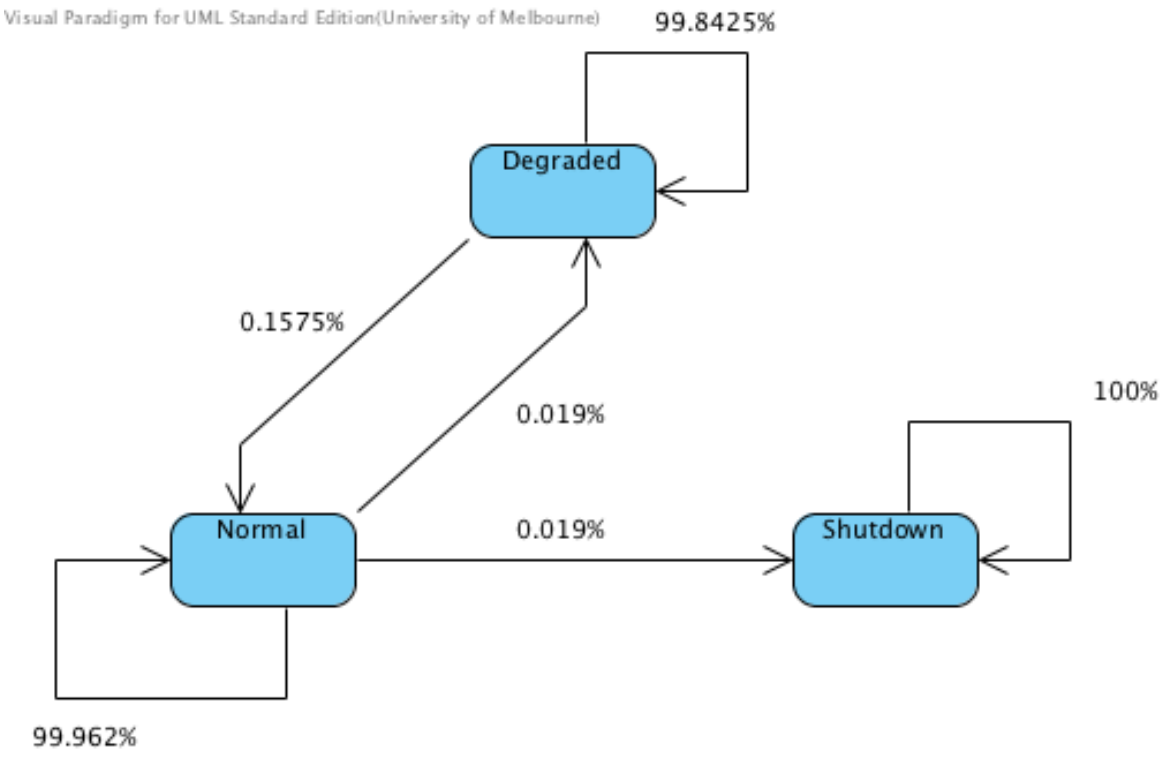
	Normal	Degraded	Shutdown
Normal	99.9616%(5218)	0.0192%(1)	0.0192%(1)
Degraded	0.2674%(1)	99.7326%(373)	0%(0)
Shutdown	0%(0)	0%(0)	100%(4406)

Average value

	Normal	Degraded	Shutdown
Normal	99.962%(13154)	0.019%(2)	0.019%(2)
Degraded	0.1575%(2)	99.8425%(1268)	0 %(0)
Shutdown	0%(0)	0%(0)	100%(5572)

Task 3

Using average value.



Task 4

In Markov model, the probability of transition to next another mode depends on only current mode.

I assume that the nuclear system starts only in normal mode. Therefore after N times multiplying of probability matrix, I will consider only normal mode row probability.

Equations.

$$P\{S(T+1) = Normal\} = P\{S(T+1) = Normal \mid S(T) = Normal\} + P\{S(T+1) = Normal \mid S(T) = Degraded\} + P\{S(T+1) = Normal \mid S(T) = Shutdown\}$$

$$P\{S(T+1) = Degraded\} = P\{S(T+1) = Degraded \mid S(T) = Normal\} + P\{S(T+1) = Degraded \mid S(T) = Degraded\} + P\{S(T+1) = Degraded \mid S(T) = Shutdown\}$$

$$P\{S(T+1) = Shutdown\} = P\{S(T+1) = Shutdown \mid S(T) = Normal\} + P\{S(T+1) = Shutdown \mid S(T) = Degraded\} + P\{S(T+1) = Shutdown \mid S(T) = Shutdown\}$$

$$\text{Tick 100} = \begin{vmatrix} 0.999748 & 0.000126 & 0.000126 \\ 0.001117 & 0.998883 & 0 \\ 0 & 0 & 100 \end{vmatrix} \wedge 100$$

$$\text{Tick 1000} = \begin{vmatrix} 0.999748 & 0.000126 & 0.000126 \\ 0.001117 & 0.998883 & 0 \\ 0 & 0 & 100 \end{vmatrix} \wedge 1000$$

$$\text{Tick 10000} = \begin{vmatrix} 0.999748 & 0.000126 & 0.000126 \\ 0.001117 & 0.998883 & 0 \\ 0 & 0 & 100 \end{vmatrix} \wedge 10000$$

Table: calculated by website (http://matrix.reshish.com/power.php)

Ticks	Normal	Degraded	Shutdown
100	99.4998%	0.24874%	0.25139%
1000	99.2527%	0.3705 %	0.3766 %
10000	99.0076%	0.4907%	0.5015%

Task 5

Build1

Ticks	Normal	Degraded	Shutdown
100	97.14%(9714)	2.52%(252)	0.34%(34)
1000	84.64%(8464)	10.55%(1055)	4.81%(481)
10000	15.33%(1533)	0.59%(59)	84.08%(8408)

Build2

Ticks	Normal	Degraded	Shutdown
100	97.05%(9705)	2.64%(264)	0.31%(31)
1000	84.69%(8469)	10.31%(1031)	5%(500)
10000	14.64%(1464)	0.58%(58)	84.78%(8478)

Task 6.

With 100 ticks the result matches to some extent. However with both 1000 ticks and 10000 ticks there are significant differences between Markov model and a test driver.

Markov model works properly if and only if the probability of transition to another mode in each tick is same regardless of tick elapsed. However the nuclear system, system under test seems that its probability of transition to shutdown mode increases in proportion to tick elapsed. The more tick elapsed, the higher probability of shutdown mode. As a result the system does not work in accordance to Markov model as a clock elapses.

Appendix A.

Task 2 Source Code

```
import java.util.Random;
/**
 * A script for interacting with the simulator.
 */
public class InteractiveDriver {
   private static final double MEAN = 1600.0f;
   private static final double VARIANCE = 100.0f;
   private static final int ITERATION = 10000;

   private static Random fRandom = new Random();
   private static int normalToNormal;
   private static int normalToDegraded;
   private static int normalToShutdown;
   private static int degradedToNormal;
   private static int degradedToDegraded;
   private static int degradedToShutdown;
   private static int shutdownToNormal;
   private static int shutdownToDegraded;
   private static int shutdownToShutdow;

   public static void run(){
      NuclearSimulator simulator = new NuclearSimulator();
      Logger logger = new Logger(true);

      normalToNormal = 0;
      normalToDegraded = 0;
      normalToShutdown = 0;
      degradedToNormal = 0;
      degradedToDegraded = 0;
      degradedToShutdown = 0;
      shutdownToNormal = 0;
```

```
        shutdownToDegraded = 0;
        shutdownToShutdow = 0;

        Mode previousMode = Mode.NORMAL;

        // 1 iteration = 1 simulated time step
        // Note: call order is important
        logger.event("Simulation begin");

        for (int iteration = 0; iteration < ITERATION;  iteration++) {

            double temperature =
    InteractiveDriver.randomNumberWithNormalDistribution();

            //send the input to the core
            simulator.update(temperature);

            logger.update();
            logger.event("Temperature: " + temperature);
            logger.event("Actuator rod at position " +
                  simulator.getActuator().getRodPosition() +
                  "; system is in " +
                  simulator.getController().getMode() + " mode\n");

            checkMode(previousMode, simulator.getController().getMode());
            previousMode = simulator.getController().getMode();
        }

        logger.event("Normal -> Normal :" + normalToNormal);
        logger.event("Normal -> Degraded:"+ normalToDegraded);
        logger.event("Normal -> Shutdown:"+ normalToShutdown);
        logger.event("Degraded -> Normal :" + degradedToNormal);
        logger.event("Degraded -> Degraded:" + degradedToDegraded);
        logger.event("Degraded -> Shutdown:" + degradedToShutdown);
        logger.event("Shutdown -> Normal :" + shutdownToNormal);
        logger.event("Shutdown -> Degraded:" + shutdownToDegraded);
        logger.event("Shutdown -> Shutdown:" + shutdownToShutdow);

        logger.event("Simulation end");
    }

    private static void checkMode(Mode previousMode, Mode currentMode) {
        if(previousMode == Mode.NORMAL){
            if(currentMode == Mode.NORMAL) normalToNormal++;
            if(currentMode == Mode.DEGRADED) normalToDegraded++;
            if(currentMode == Mode.SHUTDOWN) normalToShutdown++;
        }
        if(previousMode == Mode.DEGRADED){
            if(currentMode == Mode.NORMAL) degradedToNormal++;
```

```
            if(currentMode == Mode.DEGRADED) degradedToDegraded++;
            if(currentMode == Mode.SHUTDOWN) degradedToShutdown++;
        }
        if(previousMode == Mode.SHUTDOWN){
            if(currentMode == Mode.NORMAL) shutdownToNormal++;
            if(currentMode == Mode.DEGRADED) shutdownToDegraded++;
            if(currentMode == Mode.SHUTDOWN) shutdownToShutdow++;
        }

    }

    //Generate random temperature in accordance to operation profile.
    private static double randomNumberWithNormalDistribution(){
        return MEAN + fRandom.nextGaussian() * VARIANCE;
    }


    public static void main(String[] args){
        run();
    }
}
```

Task 5 Source Code

```
import java.util.Random;
/**
 * A script for interacting with the simulator.
 */
public class InteractiveDriver {
    private static final double MEAN = 1600.0f;
    private static final double VARIANCE = 100.0f;
    private static final int ITERATION = 10000;
    private static Random fRandom = new Random();
    private static int normalToNormal;
    private static int normalToDegraded;
    private static int normalToShutdown;

    public static void run(int tick){

        Logger logger = new Logger(true);
        normalToNormal = 0;
        normalToDegraded = 0;
        normalToShutdown = 0;

        // 1 iteration = 1 simulated time step
        // Note: call order is important
```

```
      for (int iteration = 0; iteration < ITERATION;  iteration++) {
        NuclearSimulator simulator = new NuclearSimulator();

        logger.event("Simulation begin iteration : " + iteration);
        logger.update();
        for(int i = 0 ; i < tick && simulator.getCore().isAlive(); i++){

          double temperature =
InteractiveDriver.randomNumberWithNormalDistribution();

          //send the input to the core
          simulator.update(temperature);


          logger.event("Temperature: " + temperature);
          logger.event("Actuator rod at position " +
              simulator.getActuator().getRodPosition() +
              "; system is in " +
              simulator.getController().getMode() + " mode\n");
        }
        checkMode(simulator.getController().getMode());
      }

      logger.event("Normal -> Normal :" + normalToNormal);
      logger.event("Normal -> Degraded:"+ normalToDegraded);
      logger.event("Normal -> Shutdown:"+ normalToShutdown);


      logger.event("Simulation end");
    }

    private static void checkMode(Mode currentMode) {
      if(currentMode == Mode.NORMAL) normalToNormal++;
      if(currentMode == Mode.DEGRADED) normalToDegraded++;
      if(currentMode == Mode.SHUTDOWN) normalToShutdown++;
    }

    private static double randomNumberWithNormalDistribution(){
      return MEAN + fRandom.nextGaussian() * VARIANCE;
    }


    public static void main(String[] args){
      //run(100); // 100 ticks
      //run(1000); // 1000 ticks
      run(10000); // 10000 ticks
    }
}
```