Fuzzy Logic

Project 2 Report

*Braeden Brettin
and Pierre Balinda*

Table of Contents

Table of Figures

Table of Equations

Introduction to Fuzzy Logic

Fuzzy logic systems, along with genetic algorithms and neural networks, are an important facet of advanced computational techniques. Sometimes, it is difficult to know the exact parameters and data points of a system. In these cases, programmers use what is known as "fuzzy" logic to simulate the system. For example, rather than knowing that an air conditioning system should turn on the heat when the temperature drops below 70 degrees Fahrenheit, we tell the system to turn on the heat when the temperature is "low." These "fuzzy rules," as they are called, define the behavior of the system. This approach to simulating behavior "mimics how a person would make decisions, only much faster" (Kaehler).

Fuzzy logic follows three basic steps: creating the rules, determining membership, and defuzzification. The team's project will walk through these three steps in greater detail; however, a description of each step is as follows:

1. Creating the rules: First, the parameters of the system are defined. In the case of an air conditioning system, these would be the change in temperature and the rate of change in temperature. Fuzzy rules are then created for every combination of parameters in the form of an antecedent block (If x and y) followed by a consequent block (Then z). For example, one rule for the previously-mentioned air conditioning system would be, "If the temperature has decreased and the temperature is still decreasing, then turn on the heat." Large systems, such as the one the team will create in this project, could require a plethora of rules.

2. Determining membership: The next step is to construct membership functions for each of the parameters in the system. Example membership functions for the air conditioning system are as shown below in Figure 1(Kaehler).

EXAMPLE ERROR MEMBERSHIP FUNCTION

EXAMPLE ERROR-DOT MEMBERSHIP FUNCTION
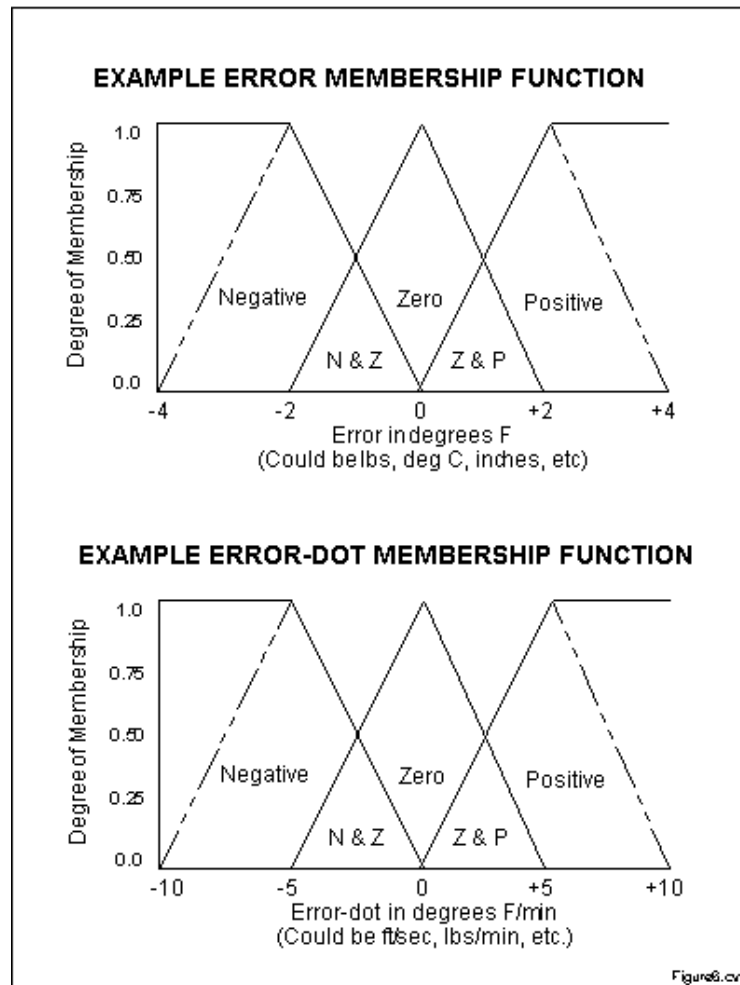
Figure 1. Example Membership Functions

Given that these membership functions provide a range of possible values for each parameter, it is now possible to match up the given values into the system with a value on this function. For example, an error in degrees of -1 degrees Fahrenheit would give a membership of 0.50 for Negative and 0.50 for Zero. The same process is applied for every membership function to give a membership for every possible linguistic variable.

3. Defuzzification: Using the calculated membership values, the rules are followed using logical AND procedures. For example, in the rule, "If the temperature has

decreased and the temperature is still decreasing, then turn on the heat," if the

membership of decreased temperature is 0.5, and the membership of decreasing

temperature is 0.25, turn on heat would result in 0.25. This procedure is followed for

every rule. The rules are then grouped by output and one of several defuzzification

methods can be used to determine a crisp numerical output. The team's chosen

defuzzification method will be addressed in greater detail later in the report.

Upon following these three steps, the program should produce one crisp output for each set of

input values to the system. In much the same way that human behavior works, this crisp output

will determine the actions needed to put the system at an ideal state. In this way, fuzzy logic can

be applied to many control systems from HVAC to fuel injection.

Brainstorming

To begin, the team spent an afternoon researching and brainstorming possible ideas for a fuzzy logic system. The team eventually decided on modeling a glucose monitoring system for diabetic patients. Given the prevalence of diabetes in the general population and the serious ramifications of not properly modulating glucose levels, the team believes this system has significant real-world applications.

Given the inexperience of the team members with glucose levels and rates, the team spent time researching ideal levels of glucose and rate of change of glucose in diabetes patients. These data points will allow us to estimate which levels and rates are too low, ideal, or too high and will serve as the range of acceptable results for the various parameters. According to Spero, an ideal glucose level for those with diabetes is about 100 mg/dL (2016). A low glucose level is about 50 mg/dL, and a high glucose level is about 150 mg/dL. According to the scholarly article written by Dunn, Eastman, and Tamada, glucose typically decreases at a rate of about -1 mg/dL/min, whereas glucose typically increases at a rate of about +1 mg/dL/min (2004).

To modulate the glucose level in the team's fictional patients, three parameters were analyzed: the current glucose level, the rate of change in glucose level, and the activity level. These three parameters are the three factors that play the largest role in regulating the glucose level of a diabetic patient. The team set three linguistic variables for the current glucose level: low, ideal, and high, three variables for the rate of change in glucose level: decreasing, constant, and increasing, and four variables for activity level: resting, minimal, intermediate, and rigorous. The team then created one-parameter fuzzy rules for each of these parameters, as shown in Figure 2, below.
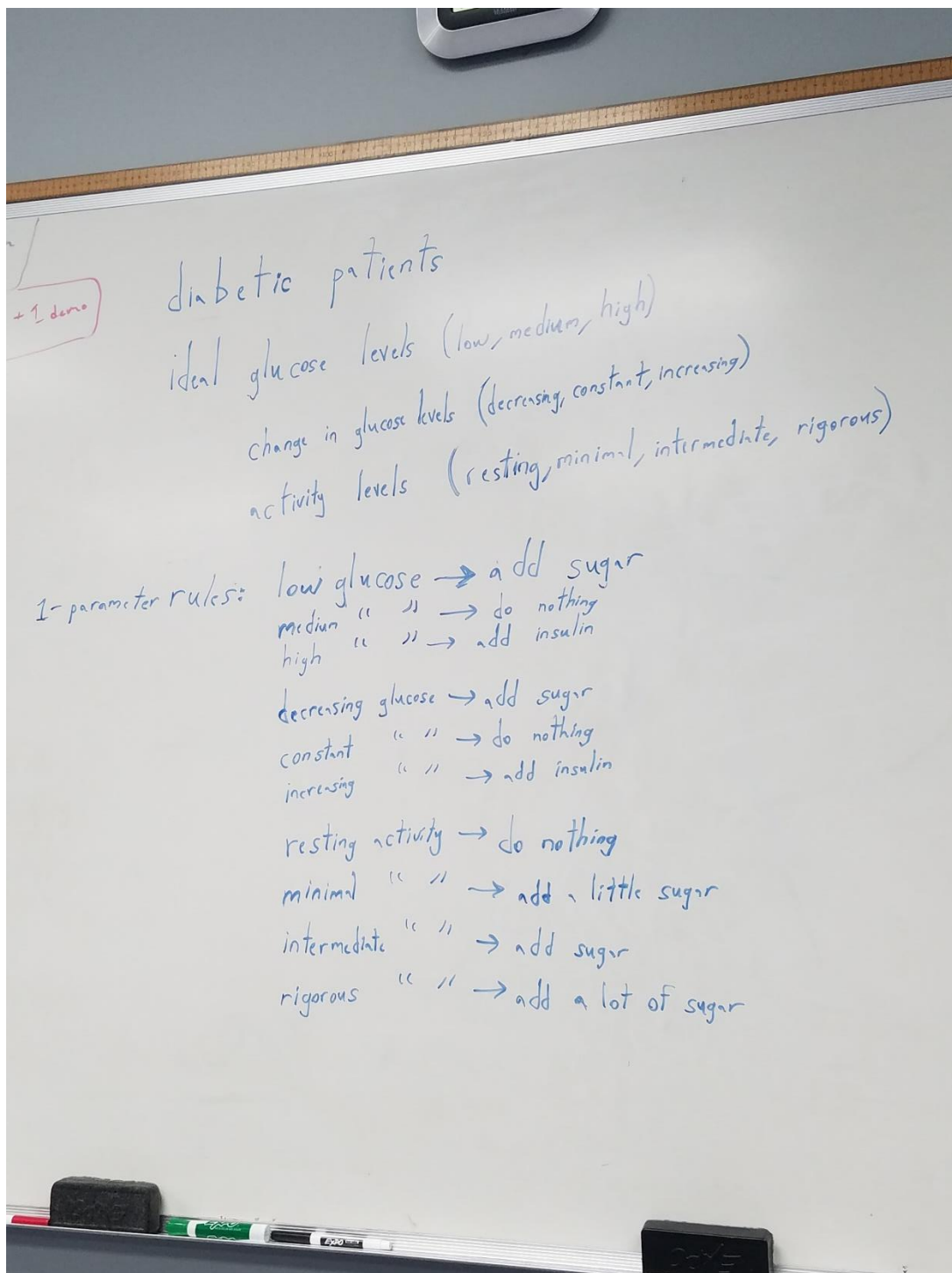
Figure 2. Brainstorming

Between this initial brainstorming session and the final project, only the number of activity

levels was changed. Rather than using four activity levels, the team decided to use two activity

levels, thus eliminating the need for qualifiers in the output (i.e. "add a lot of sugar"). After this

brainstorming session, the team still needed to accomplish three critical tasks: finishing writing

the multi-parameter rules, creating the membership functions, and determining the

defuzzification method. The team split up these tasks accordingly and proceeded to code the

program.

Developing the Rule Class

All fuzzy logic systems are run by a set of fuzzy rules, and the team's system is no different. These rules take into account all the parameters into the system and find outcomes for all possible combinations. In the case of the team's system, since three parameters will be used to define the system, the team created all the possible 1-parameter, 2-parameter, and 3-parameter rule combinations. The complete list of rules is shown below:

1. If glucose is low, then add sugar.

2. If glucose is ideal, then do nothing.

3. If glucose is high, then add insulin.

4. If glucose is decreasing, then add sugar.

5. If glucose is constant, then do nothing.

6. If glucose is increasing, then add insulin.

7. If activity is resting, then do nothing.

8. If activity is present, then add sugar.

9. If glucose is low and decreasing and activity is resting, then add sugar.

10. If glucose is low and decreasing and activity is present, then add sugar.

11. If glucose is low and constant and activity is resting, then add sugar.

12. If glucose is low and constant and activity is present, then add sugar.

13. If glucose is low and increasing and activity is resting, then do nothing.

14. If glucose is low and increasing and activity is present, then add sugar.

15. If glucose is ideal and decreasing and activity is resting, then add sugar.

16. If glucose is ideal and decreasing and activity is present, then add sugar.

17. If glucose is ideal and constant and activity is resting, then do nothing.

18. If glucose is ideal and constant and activity is present, then add sugar.

19. If glucose is ideal and increasing and activity is resting, then add insulin.

20. If glucose is ideal and increasing and activity is present, then do nothing.

21. If glucose is high and decreasing and activity is resting, then do nothing.

22. If glucose is high and decreasing and activity is present, then add sugar.

23. If glucose is high and constant and activity is resting, then add insulin.

24. If glucose is high and constant and activity is present, then do nothing.

25. If glucose is high and increasing and activity is resting, then add insulin.

26. If glucose is high and increasing and activity is present, then add insulin.

27. If glucose is low and decreasing, then add sugar.

28. If glucose is low and constant, then add sugar.

29. If glucose is low and increasing, then do nothing.

30. If glucose is ideal and decreasing, then add sugar.

31. If glucose is ideal and constant, then do nothing.

32. If glucose is ideal and increasing, then add insulin.

33. If glucose is high and decreasing, then do nothing.

34. If glucose is high and constant, then add insulin.

35. If glucose is high and increasing, then add insulin.

36. If glucose is low and activity is resting, then add sugar.

37. If glucose is low and activity is present, then add sugar.

38. If glucose is ideal and activity is resting, then do nothing.

39. If glucose is ideal and activity is present, then add sugar.

40. If glucose is high and activity is resting, then add insulin.

41. If glucose is high and activity is present, then do nothing.

42. If glucose is decreasing and activity is resting, then add sugar.

43. If glucose is decreasing and activity is present, then add sugar.

44. If glucose is constant and activity is resting, then do nothing.

45. If glucose is constant and activity is present, then add sugar.

46. If glucose is increasing and activity is resting, then add insulin.

47. If glucose is increasing and activity is present, then do nothing.

With all of the rules formulated, it was now up to the team to simulate these rules in a program. To achieve this, the team created a Rule class in Java that would model one of these rules. The constructor for this class takes in four parameters, the glucose level, the rate of change in glucose, the activity level, and the output. For example, constructing a new Rule using Rule("low", "constant", "resting", "sugar") would model Rule 11, above. Since Rules would later be sorted by output, getter methods for the output and result were added. The progress of the Rule class is shown in Figure 3, below.


```java
public class Rule
{
    String level, change, activity, output;
    double result;

    Rule(String glucose, String rate, String act, String action)
    {
        level = glucose;
        change = rate;
        activity = act;
        output = action;
    }

    String getOutput()
    {
        return output;
    }

    double getResult()
    {
        return result;
    }
}
```

Figure 3. Initial Rule Class

In order to ensure this Rule class worked as intended, a series of tests were run for each method, producing the successful output shown in Figure 4, below.
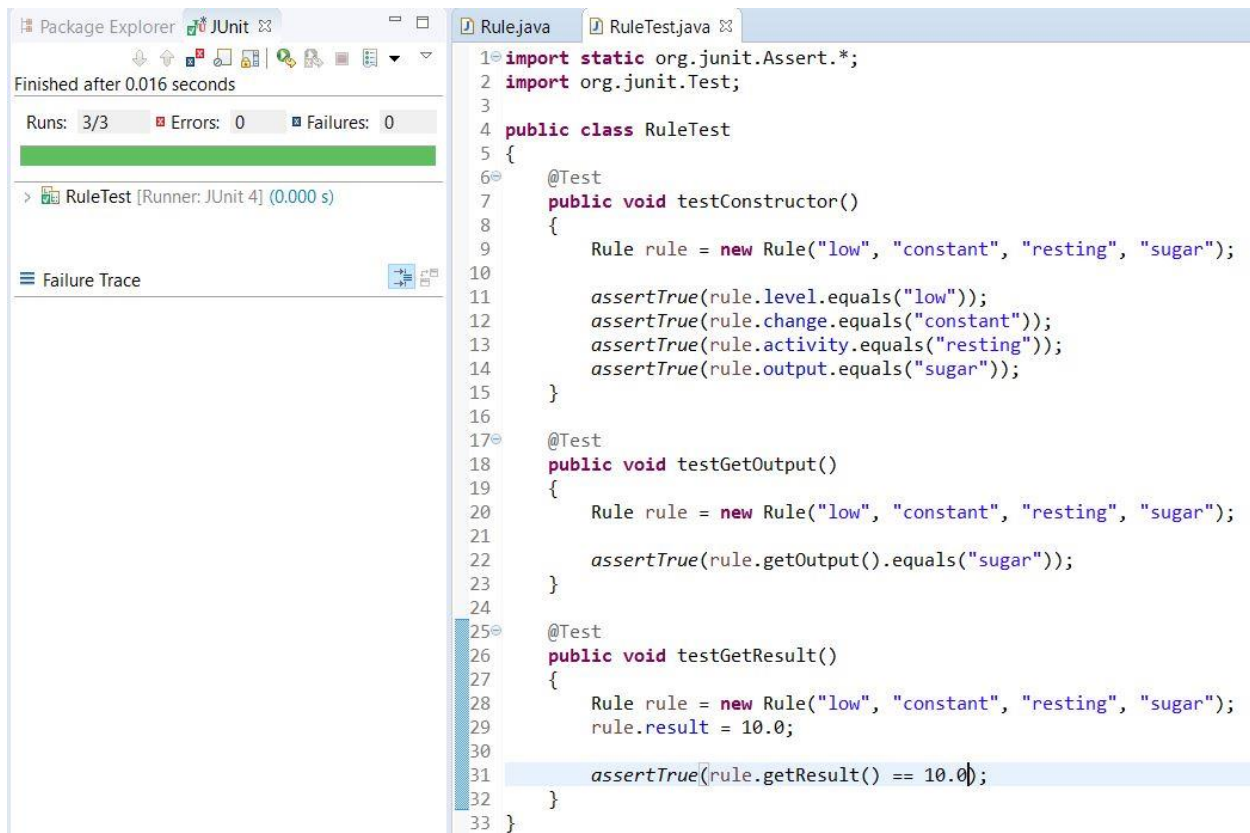
Figure 4. First Passed Test

Later in the program, it will be imperative to use memberships to return a result for each result. For example, if the input values to the system have a "low" membership of 0.1, a "constant" membership of 0.65, and a "resting" membership of 1.0, the previously mentioned rule would return a result of 0.1 using the logical AND operation. The team now wanted to create a method to calculate this result, as shown in Figure 5, below.

```
 Rule.java ☒  RuleTest.java
15⊖   void calcResult(double low, double ideal, double high, double decreasing, double constant, double increasing, double resting, double present)
16    {
17        double a = 0.0, b = 0.0, c = 0.0;
18
19        if(level.equals("low"))
20            a = low;
21        else if(level.equals("ideal"))
22            a = ideal;
23        else if(level.equals("high"))
24            a = high;
25
26        if(change.equals("decreasing"))
27            b = decreasing;
28        else if(change.equals("constant"))
29            b = constant;
30        else if(change.equals("increasing"))
31            b = increasing;
32
33        if(activity.equals("resting"))
34            c = resting;
35        else if(activity.equals("present"))
36            c = present;
37
38        result = findMin(a, b, c);
39    }
40
41    // Find the minimum of a, b, and c.
42⊖   double findMin(double a, double b, double c)
43    {
44        double min = a;
45
46        if(b < min)
47            min = b;
48
49        if(c < min)
50            min = c;
51
52        return min;
```

Figure 5. Modified Rule Class

Another series of tests was run for the calcResult and findMin methods, producing the successful

result shown in Figure 6, below.

```
 Package Explorer  JUnit ☒        □ □     Rule.java    RuleTest.java ☒
   ⬇ ⬆ ⬛ ⬛ ⬛ ◕ ◕ ▤ ▤ ▼ ▽    16
Finished after 0.015 seconds              17⊖     @Test
                                          18      public void testGetOutput()
Runs: 5/5    ▣ Errors: 0    ▣ Failures: 0 19      {
                                          20          Rule rule = new Rule("low", "constant", "resting", "sugar");
                                          21
> ⬛ RuleTest [Runner: JUnit 4] (0.000 s)  22          assertTrue(rule.getOutput().equals("sugar"));
                                          23      }
                                          24
≡ Failure Trace              ⬛ ⬛        25⊖     @Test
                                          26      public void testGetResult()
                                          27      {
                                          28          Rule rule = new Rule("low", "constant", "resting", "sugar");
                                          29          rule.result = 0.1;
                                          30
                                          31          assertTrue(rule.getResult() == 0.1);
                                          32      }
                                          33
                                          34⊖     @Test
                                          35      public void TestCalcResult()
                                          36      {
                                          37          Rule rule = new Rule("low", "constant", "resting", "sugar");
                                          38
                                          39          rule.calcResult(0.1, 0.9, 0.0, 0.35, 0.65, 0.0, 1.0, 0.0);
                                          40
                                          41          assertTrue(rule.getResult() == 0.1);
                                          42      }
                                          43
                                          44⊖     @Test
                                          45      public void TestFindMin()
                                          46      {
                                          47          Rule rule = new Rule("low", "constant", "resting", "sugar");
                                          48
                                          49          assertTrue(rule.findMin(0.3, 0.55, 0.4) == 0.3);
                                          50      }
                                          51 }
```
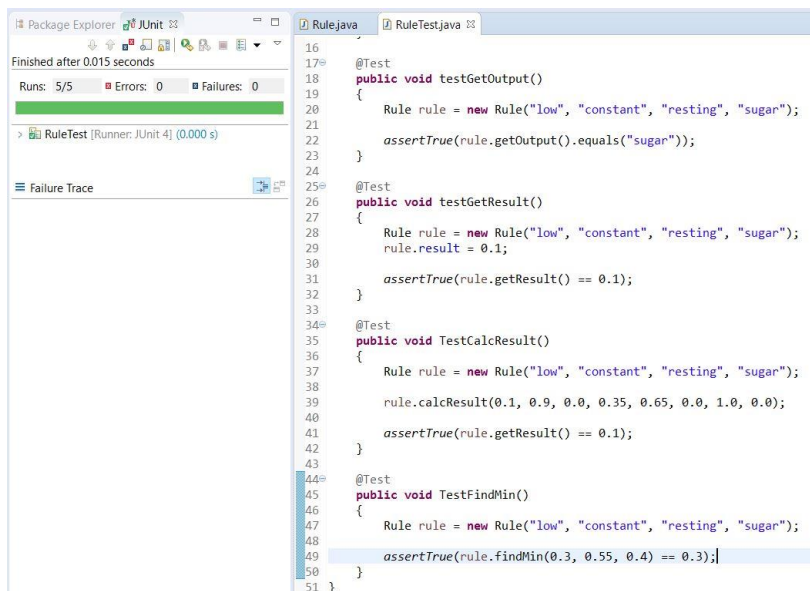
Figure 6. Second Passed Test

Developing the Fuzzy Class

With the Rule class fully constructed and working properly, it was now time to begin work on the Fuzzy class, which will contain the main method and carry out the steps of the fuzzy logic system. The first step in creating the Fuzzy class was to create an arrayList to hold all the rules for the team's fuzzy logic system. Unfortunately, there is no quick way to create all the necessary rules, so the team had to create 47 Rule objects, one for each possible rule in the system. With this work completed, the team next needed to follow the second step in any fuzzy logic system, determining membership. Any run of the program should prompt the user for three separate values, the current glucose level of the patient, the rate of change in glucose level, and the activity level. Using these values, the program should calculate the respective membership for these values in every possible parameter.

These memberships are determined by creating what is known as "membership functions." A membership function is "a graphical representation of the magnitude of participation of each input. It associates a weighting with each of the inputs that are processed, define functional overlap between inputs, and ultimately determines an output response" (Kaehler). These membership functions have several common characteristics: they are usually triangular shaped, they have centers and overlapping areas, and they use shouldering (the height is locked at maximum if an outer function (Kaehler)).

The team created three different membership functions, one for each parameter of the system. The membership function sketch for the current glucose level of the patient is as shown in Figure 7, below.
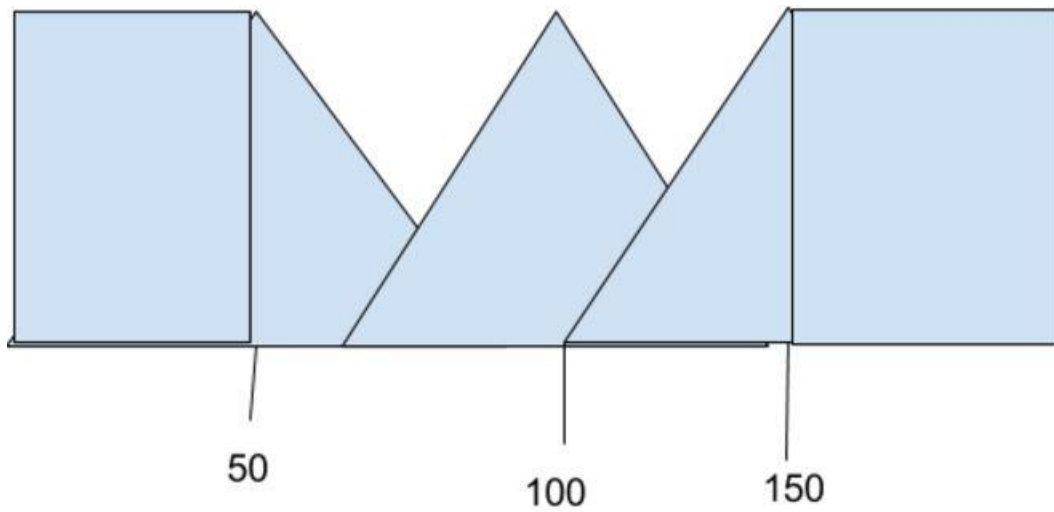
Figure 7. Glucose Level Membership Function

The membership function sketch for the rate of change in glucose is as shown in Figure 8, below.
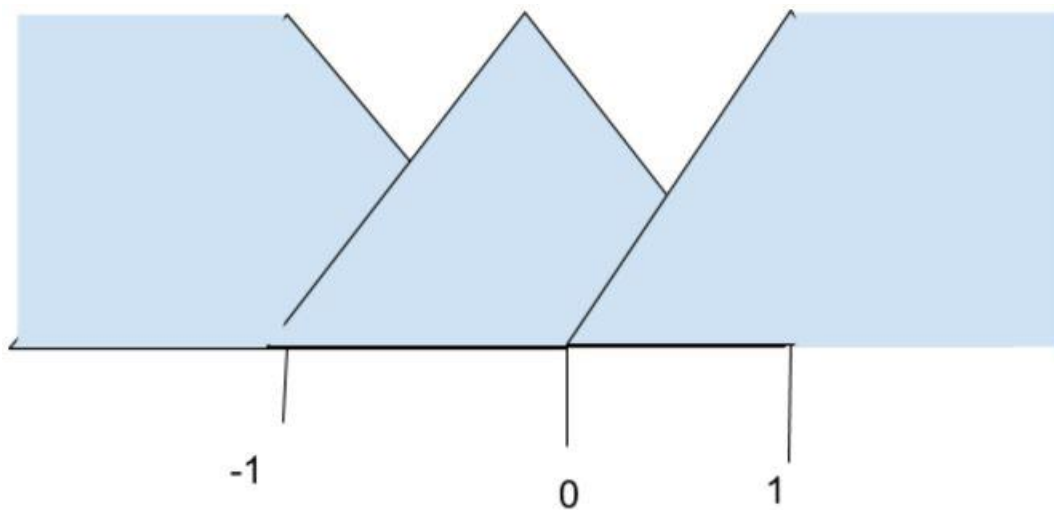

Figure 8. Rate of Change Membership Function

The membership function sketch for the activity level of the patient is as shown in Figure 9, below.

resting   present
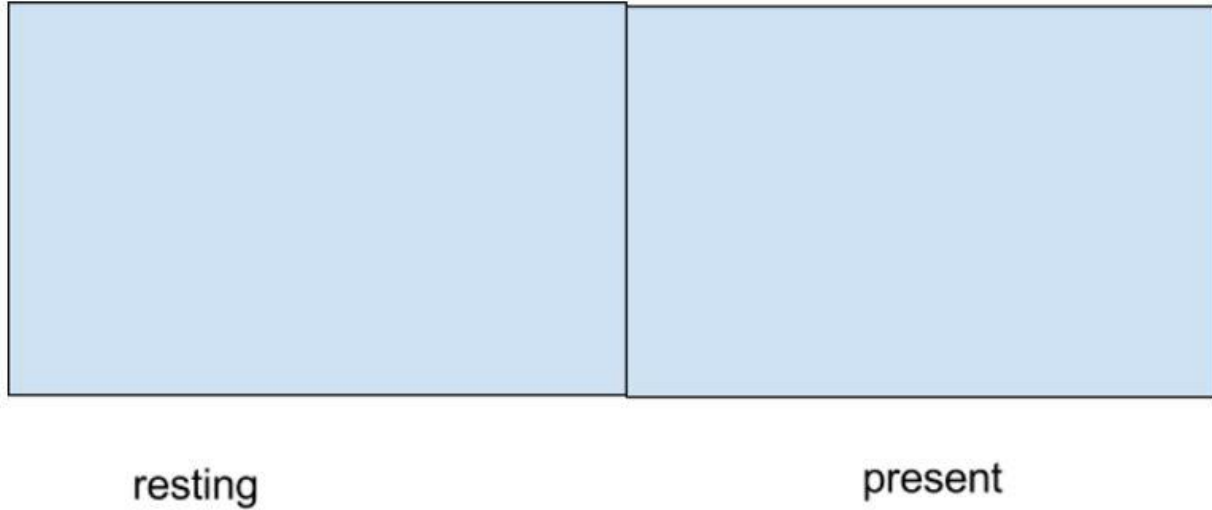
Figure 9. Activity Level Membership Function

Using these functions, the team was able to create a defined equation for each qualified parameter that can be used to numerically determine the membership of any input value in the program. The equation for low glucose membership is as shown in Equation 1, below.

$$if\ level < 50, low = 1;\ else\ low = \frac{100 - 2|level - 50|}{100}$$

Equation 1. Low Glucose Membership Equation

The equation for ideal glucose membership is as shown in Equation 2, below.

$$ideal = \frac{100 - 2|100 - level|}{100}$$

Equation 2. Ideal Glucose Membership Equation

The equation for high glucose membership is as shown in Equation 3, below.

$$if\ level > 150, high = 1;\ else\ high = \frac{100 - 2|150 - level|}{100}$$

Equation 3. High Glucose Membership Equation

The equation for decreasing glucose membership is as shown in Equation 4, below.

$$if\ rate < -1, decreasing = 1;\ else\ decreasing = |rate|$$

Equation 4. Decreasing Glucose Membership Equation

The equation for constant glucose membership is as shown in Equation 5, below.

$$constant = 1 - |rate|$$

Equation 5. Constant Glucose Membership Equation

The equation for increasing glucose membership is as shown in Equation 6, below.
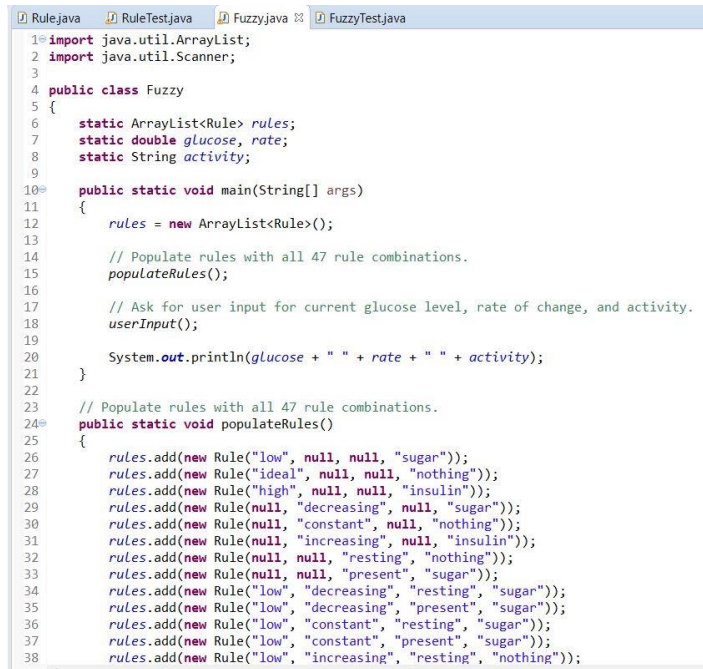
$$if\ rate > 1, increasing = 1;\ else\ increasing = rate$$

Equation 6. Increasing Glucose Membership Equation

For each of these equations, if the membership is less than 0, the membership becomes 0, because it is impossible to obtain a negative membership value.

The equation for activity level is far easier to define. The activity level is treated as a binary value. If the patient has resting activity, the resting membership is 1, and the present membership is 0. If the patient has activity present, the resting membership is 0, and the present membership is 1.

Before coding the formulas to determine membership, the team created methods for populating

the ruleset and asking for user input, as shown in Figures 10 through 12, below.

```java
Rule.java    RuleTest.java    Fuzzy.java ⊠    FuzzyTest.java
 1  import java.util.ArrayList;
 2  import java.util.Scanner;
 3
 4  public class Fuzzy
 5  {
 6      static ArrayList<Rule> rules;
 7      static double glucose, rate;
 8      static String activity;
 9
10      public static void main(String[] args)
11      {
12          rules = new ArrayList<Rule>();
13
14          // Populate rules with all 47 rule combinations.
15          populateRules();
16
17          // Ask for user input for current glucose level, rate of change, and activity.
18          userInput();
19
20          System.out.println(glucose + " " + rate + " " + activity);
21      }
22
23      // Populate rules with all 47 rule combinations.
24      public static void populateRules()
25      {
26          rules.add(new Rule("low", null, null, "sugar"));
27          rules.add(new Rule("ideal", null, null, "nothing"));
28          rules.add(new Rule("high", null, null, "insulin"));
29          rules.add(new Rule(null, "decreasing", null, "sugar"));
30          rules.add(new Rule(null, "constant", null, "nothing"));
31          rules.add(new Rule(null, "increasing", null, "insulin"));
32          rules.add(new Rule(null, null, "resting", "nothing"));
33          rules.add(new Rule(null, null, "present", "sugar"));
34          rules.add(new Rule("low", "decreasing", "resting", "sugar"));
35          rules.add(new Rule("low", "decreasing", "present", "sugar"));
36          rules.add(new Rule("low", "constant", "resting", "sugar"));
37          rules.add(new Rule("low", "constant", "present", "sugar"));
38          rules.add(new Rule("low", "increasing", "resting", "nothing"));
```

Figure 10. Initial Fuzzy Class

```java
39          rules.add(new Rule("low", "increasing", "present", "sugar"));
40          rules.add(new Rule("ideal", "decreasing", "resting", "sugar"));
41          rules.add(new Rule("ideal", "decreasing", "present", "sugar"));
42          rules.add(new Rule("ideal", "constant", "resting", "nothing"));
43          rules.add(new Rule("ideal", "constant", "present", "sugar"));
44          rules.add(new Rule("ideal", "increasing", "resting", "insulin"));
45          rules.add(new Rule("ideal", "increasing", "present", "nothing"));
46          rules.add(new Rule("high", "decreasing", "resting", "nothing"));
47          rules.add(new Rule("high", "decreasing", "present", "sugar"));
48          rules.add(new Rule("high", "constant", "resting", "insulin"));
49          rules.add(new Rule("high", "constant", "present", "nothing"));
50          rules.add(new Rule("high", "increasing", "resting", "insulin"));
51          rules.add(new Rule("high", "increasing", "present", "insulin"));
52          rules.add(new Rule("low", "decreasing", null, "sugar"));
53          rules.add(new Rule("low", "constant", null, "sugar"));
54          rules.add(new Rule("low", "increasing", null, "nothing"));
55          rules.add(new Rule("ideal", "decreasing", null, "sugar"));
56          rules.add(new Rule("ideal", "constant", null, "nothing"));
57          rules.add(new Rule("ideal", "increasing", null, "insulin"));
58          rules.add(new Rule("high", "decreasing", null, "nothing"));
59          rules.add(new Rule("high", "constant", null, "insulin"));
60          rules.add(new Rule("high", "increasing", null, "insulin"));
61          rules.add(new Rule("low", null, "resting", "sugar"));
62          rules.add(new Rule("low", null, "present", "sugar"));
63          rules.add(new Rule("ideal", null, "resting", "nothing"));
64          rules.add(new Rule("ideal", null, "present", "sugar"));
65          rules.add(new Rule("high", null, "resting", "insulin"));
66          rules.add(new Rule("high", null, "present", "nothing"));
67          rules.add(new Rule(null, "decreasing", "resting", "sugar"));
68          rules.add(new Rule(null, "decreasing", "present", "sugar"));
69          rules.add(new Rule(null, "constant", "resting", "nothing"));
70          rules.add(new Rule(null, "constant", "present", "sugar"));
71          rules.add(new Rule(null, "increasing", "resting", "insulin"));
72          rules.add(new Rule(null, "increasing", "present", "nothing"));
73      }
74
75      // Ask for user input for current glucose level, rate of change, and activity.
76      public static void userInput()
```

Figure 11. Initial Fuzzy Class
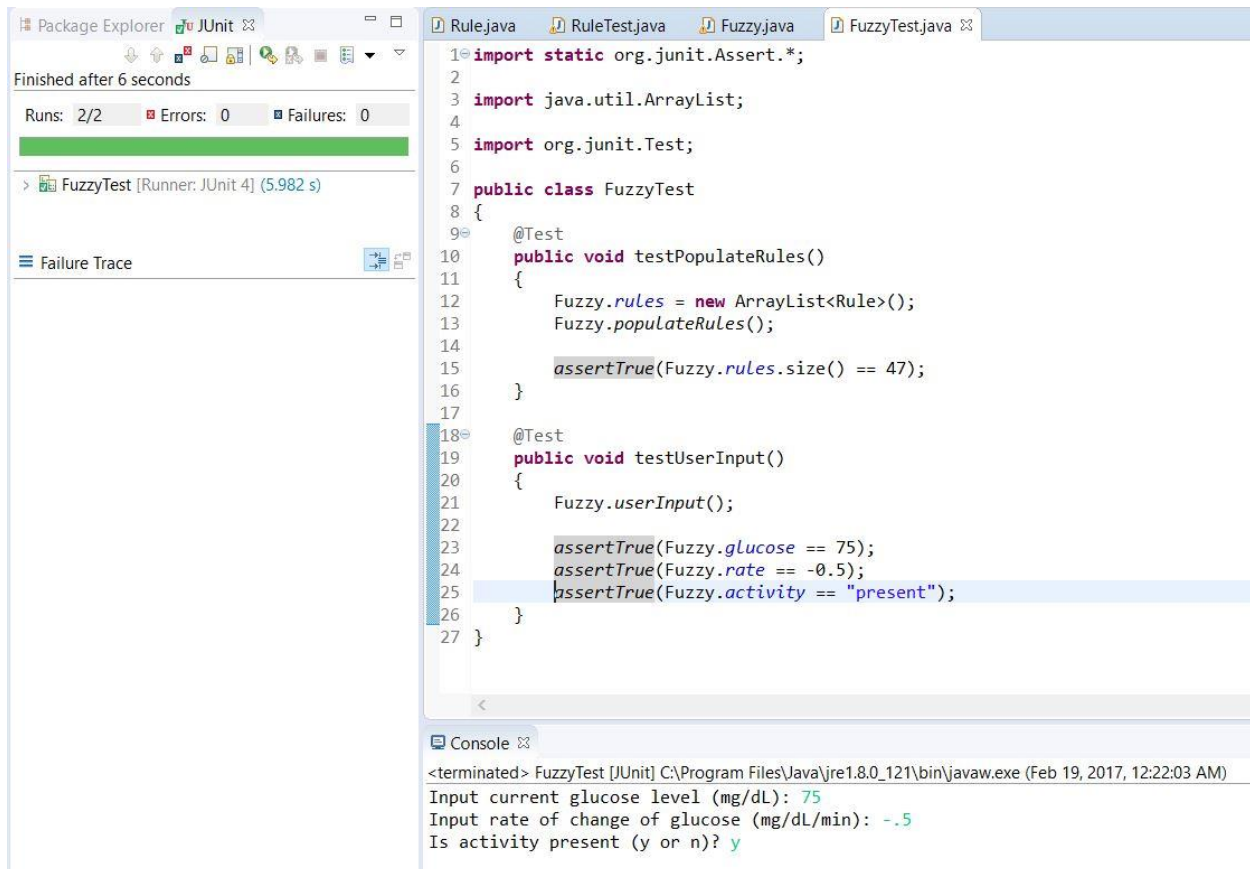
```java
77      {
78          Scanner in = new Scanner(System.in);
79
80          System.out.print("Input current glucose level (mg/dL): ");
81          glucose = in.nextDouble();
82
83          System.out.print("Input rate of change of glucose (mg/dL/min): ");
84          rate = in.nextDouble();
85
86          System.out.print("Is activity present (y or n)? ");
87          String response = in.next();
88
89          if(response.equalsIgnoreCase("y"))
90              activity = "present";
91          else if(response.equalsIgnoreCase("n"))
92              activity = "resting";
93      }
94 }
```

Figure 12. Initial Fuzzy Class

22

The team then ran tests for each of these methods, producing the successful output shown in

Figure 13, below.

Figure 13. Third Passed Test

The next step in the process of developing the Fuzzy class was to write the calcMembership

methods. These methods take in parameters such as the current glucose level, the rate of change

in glucose, and the activity level, and return the membership for each of the eight qualified

parameters using the aforementioned equations. After creating these eight methods, the code in

the Fuzzy class was as shown in Figures 14 through 16, below.

```
     Rule.java    RuleTest.java    Fuzzy.java ⌧    FuzzyTest.java
104        // Calculate the low membership.
105⊖       public static void calcLow()
106        {
107            if(glucose < 50.0)
108                low = 1.0;
109            else
110                low = (100 - 2*Math.abs(glucose-50))/100;
111
112            if(low < 0.0)
113                low = 0.0;
114        }
115
116        // Calculate the ideal membership.
117⊖       public static void calcIdeal()
118        {
119            ideal = (100 - 2*Math.abs(glucose-100))/100;
120
121            if(ideal < 0.0)
122                ideal = 0.0;
123        }
124
125        // Calculate the high membership.
126⊖       public static void calcHigh()
127        {
128            if(glucose > 150.0)
129                high = 1;
130            else
131                high = (100 - 2*Math.abs(150-glucose))/100;
132
133            if(high < 0)
134                high = 0;
135        }
136
137        // Calculate the decreasing membership.
138⊖       public static void calcDecreasing()
139        {
140            if(rate < -1)
```

Figure 14. Modified Fuzzy Class

```
     Rule.java    RuleTest.java    Fuzzy.java ⌧    FuzzyTest.java
141                decreasing = 1.0;
142            else if(rate >= 0)
143                decreasing = 0;
144            else
145                decreasing = Math.abs(rate);
146        }
147
148        // Calculate the constant membership.
149⊖       public static void calcConstant()
150        {
151            constant = 1 - Math.abs(rate);
152
153            if(constant < 0.0)
154                constant = 0.0;
155        }
156
157        // Calculate the increasing membership.
158⊖       public static void calcIncreasing()
159        {
160            if(rate > 1)
161                increasing = 1;
162            else
163                increasing = rate;
164
165            if(increasing < 0)
166                increasing = 0;
167        }
```

Figure 15. Modified Fuzzy Class

```
168
169     // Calculate the resting membership.
170⊖    public static void calcResting()
171     {
172         if(activity.equals("resting"))
173             resting = 1;
174         else
175             resting = 0;
176     }
177
178     // Calculate the present membership.
179⊖    public static void calcPresent()
180     {
181         if(activity.equals("present"))
182             present = 1;
183         else
184             present = 0;
185     }
```

Figure 16. Modified Fuzzy Class

Given the complexity of these equations, it was important for testing to examine all possible scenarios for the parameters. As such, the team's tests checked for conditions in every possible. These tests produced the successful output shown in Figures 17 through 20, below.
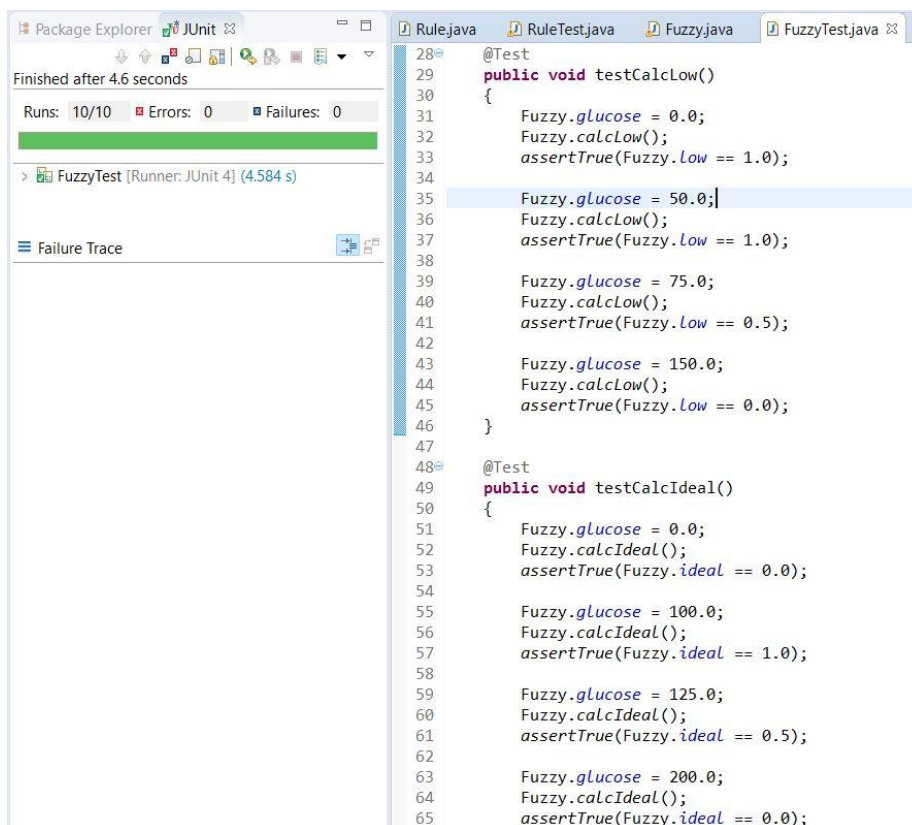
```
⊞ Package Explorer  JUnit ⊠    ▭ ▢        Rule.java    RuleTest.java    Fuzzy.java    FuzzyTest.java ⊠
          ⬇ ⬆          ▾ ▽    28⊖      @Test
Finished after 4.6 seconds           29       public void testCalcLow()
                                     30       {
Runs: 10/10   Errors: 0   Failures: 0  31           Fuzzy.glucose = 0.0;
                                     32           Fuzzy.calcLow();
                                     33           assertTrue(Fuzzy.low == 1.0);
                                     34
> FuzzyTest [Runner: JUnit 4] (4.584 s) 35           Fuzzy.glucose = 50.0;
                                     36           Fuzzy.calcLow();
                                     37           assertTrue(Fuzzy.low == 1.0);
                                     38
≡ Failure Trace                      39           Fuzzy.glucose = 75.0;
                                     40           Fuzzy.calcLow();
                                     41           assertTrue(Fuzzy.low == 0.5);
                                     42
                                     43           Fuzzy.glucose = 150.0;
                                     44           Fuzzy.calcLow();
                                     45           assertTrue(Fuzzy.low == 0.0);
                                     46       }
                                     47
                                     48⊖      @Test
                                     49       public void testCalcIdeal()
                                     50       {
                                     51           Fuzzy.glucose = 0.0;
                                     52           Fuzzy.calcIdeal();
                                     53           assertTrue(Fuzzy.ideal == 0.0);
                                     54
                                     55           Fuzzy.glucose = 100.0;
                                     56           Fuzzy.calcIdeal();
                                     57           assertTrue(Fuzzy.ideal == 1.0);
                                     58
                                     59           Fuzzy.glucose = 125.0;
                                     60           Fuzzy.calcIdeal();
                                     61           assertTrue(Fuzzy.ideal == 0.5);
                                     62
                                     63           Fuzzy.glucose = 200.0;
                                     64           Fuzzy.calcIdeal();
                                     65           assertTrue(Fuzzy.ideal == 0.0);
```
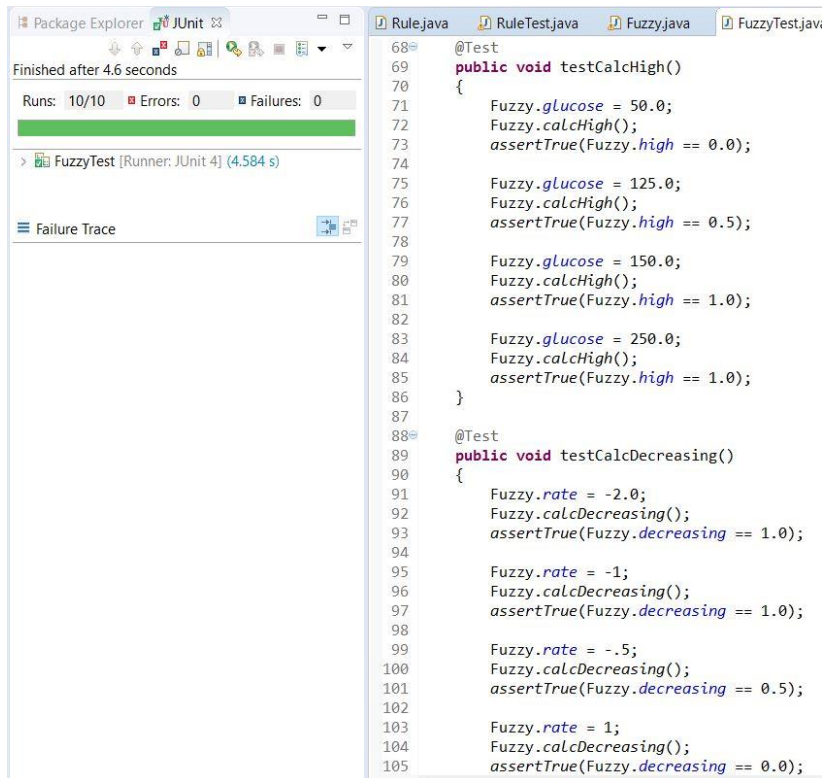
Figure 17. Fourth Passed Test

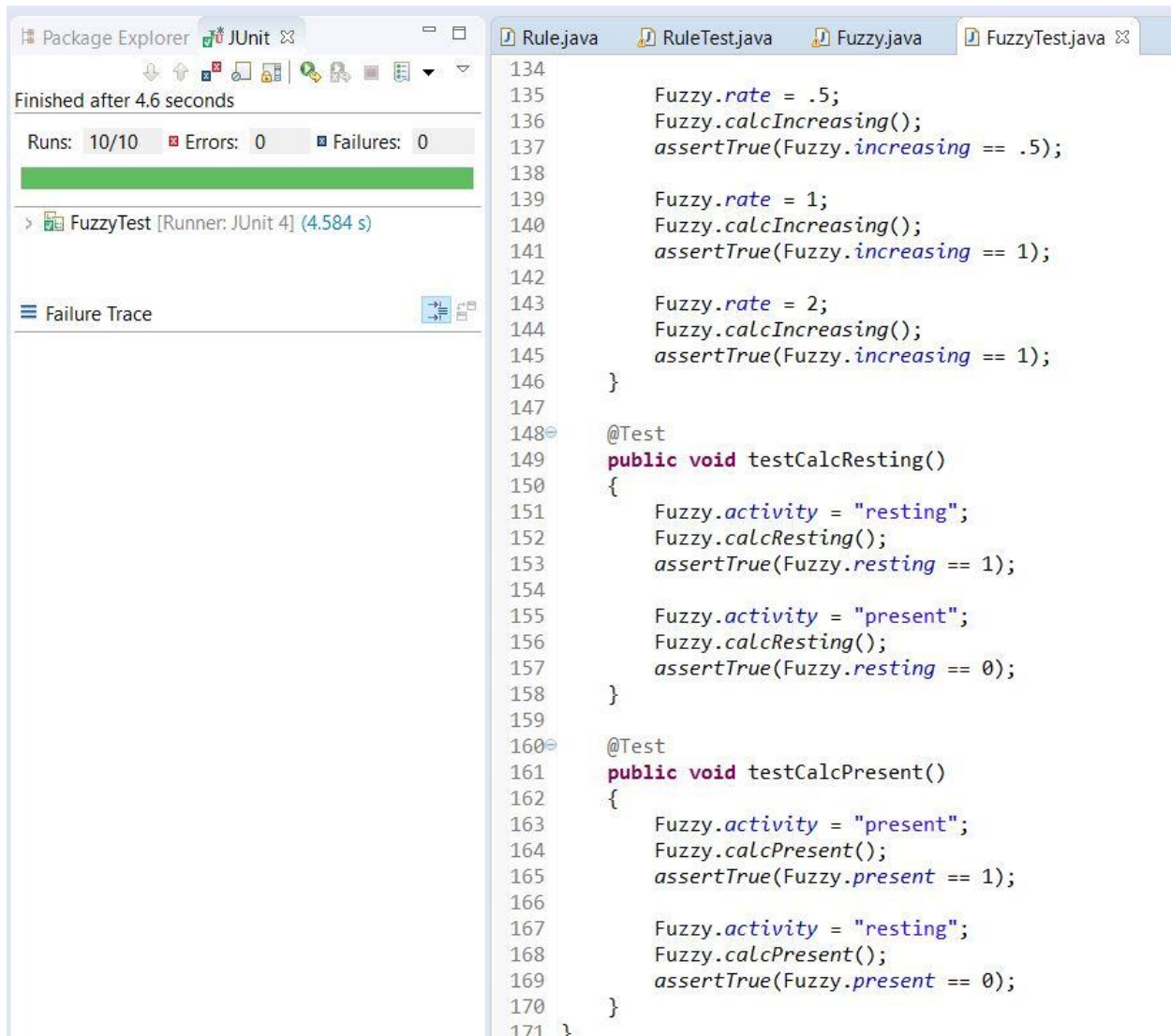Figure 18. Fourth Passed Test
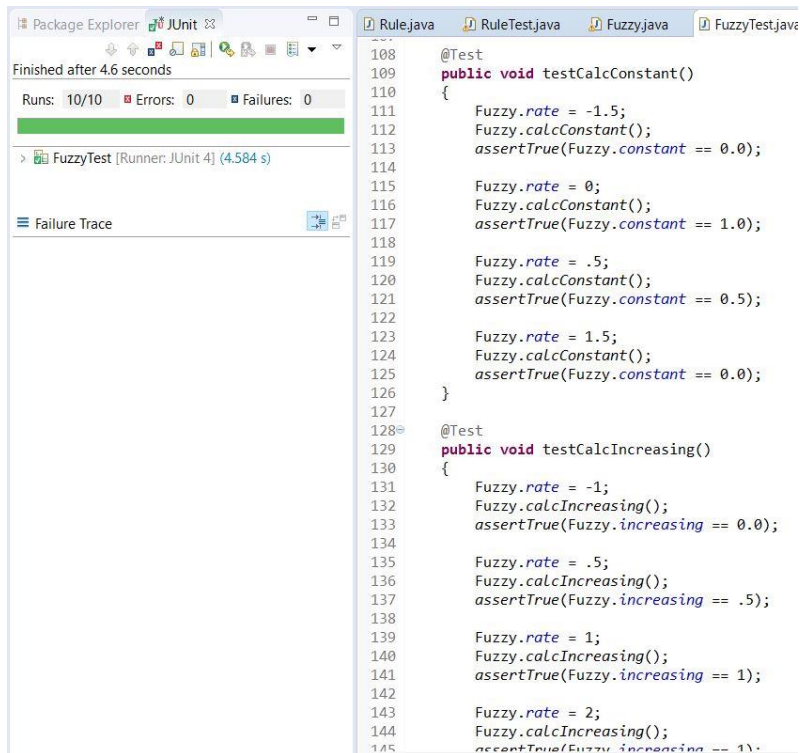
Figure 19. Fourth Passed Test

```
134
135            Fuzzy.rate = .5;
136            Fuzzy.calcIncreasing();
137            assertTrue(Fuzzy.increasing == .5);
138
139            Fuzzy.rate = 1;
140            Fuzzy.calcIncreasing();
141            assertTrue(Fuzzy.increasing == 1);
142
143            Fuzzy.rate = 2;
144            Fuzzy.calcIncreasing();
145            assertTrue(Fuzzy.increasing == 1);
146        }
147
148⊖      @Test
149        public void testCalcResting()
150        {
151            Fuzzy.activity = "resting";
152            Fuzzy.calcResting();
153            assertTrue(Fuzzy.resting == 1);
154
155            Fuzzy.activity = "present";
156            Fuzzy.calcResting();
157            assertTrue(Fuzzy.resting == 0);
158        }
159
160⊖      @Test
161        public void testCalcPresent()
162        {
163            Fuzzy.activity = "present";
164            Fuzzy.calcPresent();
165            assertTrue(Fuzzy.present == 1);
166
167            Fuzzy.activity = "resting";
168            Fuzzy.calcPresent();
169            assertTrue(Fuzzy.present == 0);
170        }
171 }
```

Figure 20. Fourth Passed Test

After coding and testing all levels of membership calculations, the team proceed to developing the defuzzification process of the system. The team followed the aforementioned tutorial's example to setup this process. The first step of any defuzzification process is to create the output membership set and functions and determine their interval. The output membership set was "Add sugar", "Do nothing", and "Add insulin", equation 7 –the root-sum-square(RSS)— was used to calculate their membership function intervals for each patient. "Add sugar" will

cover output values between -100 and 0, "Nothing" will cover 0, and "Add insulin" will cover 0 to 100.

$Sugar = \sqrt{(\sum_{k=0}^{n} n^2)}$ where n is the membership degree of a rule, whose action is sugar

$Nothing = \sqrt{(\sum_{k=0}^{n} n^2)}$ where n is the membership degree of a rule, whose action is nothing

$Insulin = \sqrt{(\sum_{k=0}^{n} n^2)}$ where n is the membership degree of a rule, whose action is insulin

Equation 7: Computation of intervals using RSS

The team then proceed to computing the crisps output value using the fuzzy centroid algorithm, as shown in equation 8.
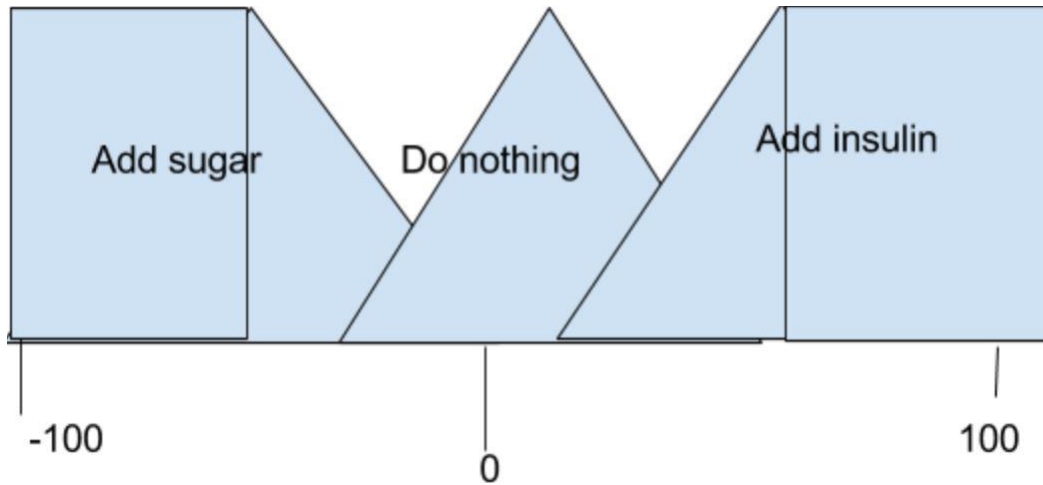
Figure 21: Output membership function

$$Output = \frac{(\text{sugar} * -100) + (\text{nothing} * 0) + (insulin * 100)}{\text{sugar} + \text{nothing} + \text{insulin}}$$

Equation 8: Crisp value equation

After the developing these two equations, the team proceed to creating unit tests and methods that carry out the computations, as seen in Figure 22 and 23.

```
69     @Test
70     public void testCalcInterval()
71     {
72         Fuzzy.populateRules();
73         Fuzzy.userInput();
74         Fuzzy.calcIntervals();
75         assertTrue(Fuzzy.output == -100);
76     }
```

Figure 22: TestCalcInterval unit test

```
197
198    //determining the output levels of membership
199    public static void calcIntervals()
200    {
201        calcMembership();
202        for(int i=0; i< rules.size(); i++)
203        {
204            rules.get(i).calcResult( low,  ideal,  high,  decreasing,  constant,  increasing,  resting,  present);
205            //sugar/insulin/nothing = sum of respective squares
206            if( rules.get(i).getOutput() == "sugar")
207                sugar +=Math.pow(rules.get(i).getResult(), 2);
208            else if (rules.get(i).getOutput() == "insulin")
209                insulin += Math.pow(rules.get(i).getResult(), 2);
210            else
211                nothing += Math.pow(rules.get(i).getResult(),2);
212        }
213        sugar = Math.sqrt(sugar);
214        insulin = Math.sqrt(insulin);
215        nothing = Math.sqrt(nothing);
216        if(sugar + nothing + insulin != 0)  //Avoid dividing by 0
217            output = ((-100 * sugar) +  (0*nothing) + (100*insulin))/(sugar + nothing + insulin);
218        else
219            output = 0;
220    }
```

Figure 23: calcIntervals method

The method to interpret the output data was tested and written as seen below in Figure 23 and 24.

The interpretation was based off the output membership function figure. The test was successful

as shown in Figure 25.

```
177    @Test
178    public void testInterpret()
179    {
180        Fuzzy.populateRules();
181        Fuzzy.userInput();
182        Fuzzy.calcIntervals();
183        assertTrue(Fuzzy.interpret() == "Add sugar");
184    }
```

Figure 24: testInterpret() unit test

```
221    public static String interpret()
222    {
223        if(output > 0)
224            return "Add insulin";
225        if (output < 0)
226            return "Add sugar";
227        else
228            return "Do nothing";
229    }
```

Figure 25: Interpret method

∨ FuzzyTest [Runner: JUnit 4] (0.038 s)
    testCalcIdeal (0.000 s)
    testCalcConstant (0.000 s)
    testCalcPresent (0.000 s)
    testCalcLow (0.000 s)
    testCalcIncreasing (0.000 s)
    testCalcDecreasing (0.000 s)
    testCalcHigh (0.000 s)
    testCalcInterval (0.036 s)
    testInterpret (0.002 s)
    testCalcResting (0.000 s)
    testPopulateRules (0.000 s)

Figure 26: Successful unit tests

References

Dunn, T. C., Eastman, R. C., & Tamada, J. A. (Sep. 2004). Rates of Glucose Change Measured by Blood Glucose Meter and the GlucoWatch Biographer During Day, Night, and Around Mealtimes. *Diabetes Care*, 27(9), 2161-2165. Retrieved from http://care.diabetesjournals.org/content/27/9/2161

Engelbrecht, A. P. (n.d.). *Computational Intelligence: An Introduction* (2nd ed.).

Kaehler, S. D. (n.d.). *Fuzzy Logic Tutorial*. Retrieved from http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html

Spero, D. (Jan. 13, 2016). *What is a Normal Blood Sugar Level?* Retrieved from https://www.diabetesselfmanagement.com/blog/what-is-a-normal-blood-sugar-level/

(n.d.). *Fuzzy Logic*. Retrieved from http://www.rpi.edu/dept/ecse/mps/Fuzzy_Logic.pdf

Appendix

A. GitHub repository

B. Fuzzy.java

```
import java.util.ArrayList;

import java.util.Scanner;


public class Fuzzy

{

    static ArrayList<Rule> rules =new ArrayList<Rule>();

    static double glucose, rate;

    static String activity;

    static double low, ideal, high, decreasing, constant, increasing, resting, present, output,

insulin=0, sugar=0, nothing=0;


    public static void main(String[] args)

    {

            //rules = new ArrayList<Rule>();
```

```java
        // Populate rules with all 47 rule combinations.

        populateRules();


        // Ask for user input for current glucose level, rate of change, and activity.

        userInput();


        // Calculate the memberships for the various qualified parameters.

        calcLow();

        calcIdeal();

        calcHigh();

        calcDecreasing();

        calcConstant();

        calcIncreasing();

        calcResting();

        calcPresent();

        calcIntervals();

        System.out.println("\nOutput= " + output);

        System.out.println("Recommended action: " + interpret());

    }


// Populate rules with all 47 rule combinations.

public static void populateRules()

{
```

```
rules.add(new Rule("low", null, null, "sugar"));

rules.add(new Rule("ideal", null, null, "nothing"));

rules.add(new Rule("high", null, null, "insulin"));

rules.add(new Rule(null, "decreasing", null, "sugar"));

rules.add(new Rule(null, "constant", null, "nothing"));

rules.add(new Rule(null, "increasing", null, "insulin"));

rules.add(new Rule(null, null, "resting", "nothing"));

rules.add(new Rule(null, null, "present", "sugar"));

rules.add(new Rule("low", "decreasing", "resting", "sugar"));

rules.add(new Rule("low", "decreasing", "present", "sugar"));

rules.add(new Rule("low", "constant", "resting", "sugar"));

rules.add(new Rule("low", "constant", "present", "sugar"));

rules.add(new Rule("low", "increasing", "resting", "nothing"));

rules.add(new Rule("low", "increasing", "present", "sugar"));

rules.add(new Rule("ideal", "decreasing", "resting", "sugar"));

rules.add(new Rule("ideal", "decreasing", "present", "sugar"));

rules.add(new Rule("ideal", "constant", "resting", "nothing"));

rules.add(new Rule("ideal", "constant", "present", "sugar"));

rules.add(new Rule("ideal", "increasing", "resting", "insulin"));

rules.add(new Rule("ideal", "increasing", "present", "nothing"));

rules.add(new Rule("high", "decreasing", "resting", "nothing"));

rules.add(new Rule("high", "decreasing", "present", "sugar"));

rules.add(new Rule("high", "constant", "resting", "insulin"));
```

```java
rules.add(new Rule("high", "constant", "present", "nothing"));

rules.add(new Rule("high", "increasing", "resting", "insulin"));

rules.add(new Rule("high", "increasing", "present", "insulin"));

rules.add(new Rule("low", "decreasing", null, "sugar"));

rules.add(new Rule("low", "constant", null, "sugar"));

rules.add(new Rule("low", "increasing", null, "nothing"));

rules.add(new Rule("ideal", "decreasing", null, "sugar"));

rules.add(new Rule("ideal", "constant", null, "nothing"));

rules.add(new Rule("ideal", "increasing", null, "insulin"));

rules.add(new Rule("high", "decreasing", null, "nothing"));

rules.add(new Rule("high", "constant", null, "insulin"));

rules.add(new Rule("high", "increasing", null, "insulin"));

rules.add(new Rule("low", null, "resting", "sugar"));

rules.add(new Rule("low", null, "present", "sugar"));

rules.add(new Rule("ideal", null, "resting", "nothing"));

rules.add(new Rule("ideal", null, "present", "sugar"));

rules.add(new Rule("high", null, "resting", "insulin"));

rules.add(new Rule("high", null, "present", "nothing"));

rules.add(new Rule(null, "decreasing", "resting", "sugar"));

rules.add(new Rule(null, "decreasing", "present", "sugar"));

rules.add(new Rule(null, "constant", "resting", "nothing"));

rules.add(new Rule(null, "constant", "present", "sugar"));

rules.add(new Rule(null, "increasing", "resting", "insulin"));
```

```java
        rules.add(new Rule(null, "increasing", "present", "nothing"));

}


// Ask for user input for current glucose level, rate of change, and activity.

public static void userInput()

{

        Scanner in = new Scanner(System.in);

        System.out.print("Input current glucose level (mg/dL): ");

        glucose = in.nextDouble();

        System.out.print("Input rate of change of glucose (mg/dL/min): ");

        rate = in.nextDouble();

        System.out.print("Is activity present (y or n)? ");

        String response = in.next();

        if(response==("y"))

                activity = "present";

        else if(response==("n"))

                activity = "resting";

}


// Calculate the low membership.

public static void calcLow()

{

        if(glucose < 50.0)
```

```java
                low = 1.0;

        else

                low = (100 - 2*Math.abs(glucose-50))/100;


        if(low < 0.0)

                low = 0.0;

}


// Calculate the ideal membership.

public static void calcIdeal()

{

        ideal = (100 - 2*Math.abs(glucose-100))/100;


        if(ideal < 0.0)

                ideal = 0.0;

}


// Calculate the high membership.

public static void calcHigh()

{

        if(glucose > 150.0)

                high = 1;

        else
```

```
                  high = (100 - 2*Math.abs(150-glucose))/100;


       if(high < 0)

              high = 0;

}


// Calculate the decreasing membership.

public static void calcDecreasing()

{

       if(rate < -1)

              decreasing = 1.0;

       else if(rate >= 0)

              decreasing = 0;

       else

              decreasing = Math.abs(rate);

}


// Calculate the constant membership.

public static void calcConstant()

{

       constant = 1 - Math.abs(rate);


       if(constant < 0.0)
```

```java
                constant = 0.0;

}


// Calculate the increasing membership.

public static void calcIncreasing()

{

        if(rate > 1)

                increasing = 1;

        else

                increasing = rate;


        if(increasing < 0)

                increasing = 0;

}


// Calculate the resting membership.

public static void calcResting()

{

        if(activity==("resting"))

                resting = 1;

        else

                resting = 0;

}
```

```java
// Calculate the present membership.

public static void calcPresent()

{

        if(activity==("present"))

                present = 1;

        else

                present = 0;

}

//calculates all memberships

public static void calcMembership()

{

        calcLow();

        calcIdeal();

        calcHigh();

        calcDecreasing();

        calcConstant();

        calcIncreasing();

        calcResting();

        calcPresent();

}


//determining the output levels of membership
```

```java
public static void calcIntervals()

{

        calcMembership();

        for(int i=0; i< rules.size(); i++)

        {

                rules.get(i).calcResult( low, ideal, high, decreasing, constant,

increasing, resting, present);

                //sugar/insulin/nothing = sum of respective squares

                if( rules.get(i).getOutput() == "sugar")

                        sugar +=Math.pow(rules.get(i).getResult(), 2);

                else if (rules.get(i).getOutput() == "insulin")

                        insulin += Math.pow(rules.get(i).getResult(), 2);

                else

                        nothing += Math.pow(rules.get(i).getResult(),2);

        }

        sugar = Math.sqrt(sugar);

        insulin = Math.sqrt(insulin);

        nothing = Math.sqrt(nothing);

        if(sugar + nothing + insulin != 0)        //Avoid dividing by 0

                output = ((-100 * sugar) +  (0*nothing) + (100*insulin))/(sugar + nothing

+ insulin);

        else

                output = 0;
```

```java
        }

        public static String interpret()

        {

                if(output > 0)

                        return "Add insulin";

                if (output < 0)

                        return "Add sugar";

                else

                        return "Do nothing";

        }

    }
```

C. FuzzyTest.java

```java
import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Test;

public class FuzzyTest

{

    @Test

    public void testPopulateRules()

    {
```

```java
        Fuzzy.rules = new ArrayList<Rule>();

        Fuzzy.populateRules();


        assertTrue(Fuzzy.rules.size() == 47);

}


@Test

public void testUserInput()

{

        Fuzzy.userInput();

        assertTrue(Fuzzy.glucose == 75.0);

        assertTrue(Fuzzy.rate == -0.5);

        assertTrue(Fuzzy.activity == "present");

}


@Test

public void testCalcLow()

{

        Fuzzy.glucose = 0.0;

        Fuzzy.calcLow();

        assertTrue(Fuzzy.low == 1.0);

        Fuzzy.glucose = 50.0;

        Fuzzy.calcLow();
```

```java
        assertTrue(Fuzzy.low == 1.0);


        Fuzzy.glucose = 75.0;

        Fuzzy.calcLow();

        assertTrue(Fuzzy.low == 0.5);


        Fuzzy.glucose = 150.0;

        Fuzzy.calcLow();

        assertTrue(Fuzzy.low == 0.0);
}


@Test

public void testCalcIdeal()

{
        Fuzzy.glucose = 0.0;

        Fuzzy.calcIdeal();

        assertTrue(Fuzzy.ideal == 0.0);


        Fuzzy.glucose = 100.0;

        Fuzzy.calcIdeal();

        assertTrue(Fuzzy.ideal == 1.0);


        Fuzzy.glucose = 125.0;
```

```java
        Fuzzy.calcIdeal();

        assertTrue(Fuzzy.ideal == 0.5);


        Fuzzy.glucose = 200.0;

        Fuzzy.calcIdeal();

        assertTrue(Fuzzy.ideal == 0.0);

}


@Test

public void testCalcHigh()

{

        Fuzzy.glucose = 50.0;

        Fuzzy.calcHigh();

        assertTrue(Fuzzy.high == 0.0);


        Fuzzy.glucose = 125.0;

        Fuzzy.calcHigh();

        assertTrue(Fuzzy.high == 0.5);


        Fuzzy.glucose = 150.0;

        Fuzzy.calcHigh();

        assertTrue(Fuzzy.high == 1.0);
```

```java
        Fuzzy.glucose = 250.0;

        Fuzzy.calcHigh();

        assertTrue(Fuzzy.high == 1.0);

}


@Test

public void testCalcDecreasing()

{

        Fuzzy.rate = -2.0;

        Fuzzy.calcDecreasing();

        assertTrue(Fuzzy.decreasing == 1.0);


        Fuzzy.rate = -1;

        Fuzzy.calcDecreasing();

        assertTrue(Fuzzy.decreasing == 1.0);


        Fuzzy.rate = -.5;

        Fuzzy.calcDecreasing();

        assertTrue(Fuzzy.decreasing == 0.5);


        Fuzzy.rate = 1;

        Fuzzy.calcDecreasing();

        assertTrue(Fuzzy.decreasing == 0.0);
```

```
}


@Test

public void testCalcConstant()

{

        Fuzzy.rate = -1.5;

        Fuzzy.calcConstant();

        assertTrue(Fuzzy.constant == 0.0);


        Fuzzy.rate = 0;

        Fuzzy.calcConstant();

        assertTrue(Fuzzy.constant == 1.0);


        Fuzzy.rate = .5;

        Fuzzy.calcConstant();

        assertTrue(Fuzzy.constant == 0.5);


        Fuzzy.rate = 1.5;

        Fuzzy.calcConstant();

        assertTrue(Fuzzy.constant == 0.0);

}


@Test
```

```java
public void testCalcIncreasing()

{

        Fuzzy.rate = -1;

        Fuzzy.calcIncreasing();

        assertTrue(Fuzzy.increasing == 0.0);


        Fuzzy.rate = .5;

        Fuzzy.calcIncreasing();

        assertTrue(Fuzzy.increasing == .5);


        Fuzzy.rate = 1;

        Fuzzy.calcIncreasing();

        assertTrue(Fuzzy.increasing == 1);


        Fuzzy.rate = 2;

        Fuzzy.calcIncreasing();

        assertTrue(Fuzzy.increasing == 1);

}


@Test

public void testCalcResting()

{

        Fuzzy.activity = "resting";
```

```java
        Fuzzy.calcResting();

        assertTrue(Fuzzy.resting == 1);


        Fuzzy.activity = "present";

        Fuzzy.calcResting();

        assertTrue(Fuzzy.resting == 0);

}


@Test

public void testCalcPresent()

{

        Fuzzy.activity = "present";

        Fuzzy.calcPresent();

        assertTrue(Fuzzy.present == 1);


        Fuzzy.activity = "resting";

        Fuzzy.calcPresent();

        assertTrue(Fuzzy.present == 0);

}

@Test

public void testCalcInterval()

{

        Fuzzy.populateRules();
```

```
            Fuzzy.userInput();

            Fuzzy.calcIntervals();

            assertTrue(Fuzzy.output == -100);

        }

        @Test

        public void testInterpret()

        {

                Fuzzy.populateRules();

                Fuzzy.userInput();

                Fuzzy.calcIntervals();

                assertTrue(Fuzzy.interpret() == "Add sugar");

        }

}
```

D. Rule.java

```java
public class Rule
{
    String level, change, activity, output;
    double result;

    Rule(String glucose, String rate, String act, String action)
    {
        level = glucose;
        change = rate;
        activity = act;
        output = action;
    }

    // The parameters are the membership values for each level, change, and
activity.
    void calcResult(double low, double ideal, double high, double
decreasing, double constant, double increasing, double resting, double
present)
    {
        double a = 0.0, b = 0.0, c = 0.0;

        if(level==("low"))
                a = low;
```

```java
            else if(level==("ideal"))
                    a = ideal;
            else if(level==("high"))
                    a = high;

            if(change==("decreasing"))
                    b = decreasing;
            else if(change==("constant"))
                    b = constant;
            else if(change==("increasing"))
                    b = increasing;

            if(activity==("resting"))
                    c = resting;
            else if(activity==("present"))
                    c = present;

            result = findMin(a, b, c);
    }

    // Find the minimum of a, b, and c.
    double findMin(double a, double b, double c)
    {
            double min = a;

            if(b < min)
                    min = b;

            if(c < min)
                    min = c;

            return min;
    }

    String getOutput()
    {
            return output;
    }

    double getResult()
    {
            return result;
    }
}
```

E. RuleTest.java

import static org.junit.Assert.*;

import org.junit.Test;

public class RuleTest

```java
{
    @Test
    public void testRule()
    {
        Rule rule = new Rule("low", "constant", "resting", "sugar");

        assertTrue(rule.level.equals("low"));
        assertTrue(rule.change.equals("constant"));
        assertTrue(rule.activity.equals("resting"));
        assertTrue(rule.output.equals("sugar"));

        Rule rule2 = new Rule("low", null, null, "sugar");

        assertTrue(rule2.level.equals("low"));
        assertTrue(rule2.change == null);
        assertTrue(rule2.activity == null);
        assertTrue(rule2.output.equals("sugar"));
    }

    @Test
    public void testGetOutput()
    {
        Rule rule = new Rule("low", "constant", "resting", "sugar");
```

```java
        assertTrue(rule.getOutput().equals("sugar"));

}


@Test

public void testGetResult()

{

        Rule rule = new Rule("low", "constant", "resting", "sugar");

        rule.result = 0.1;


        assertTrue(rule.getResult() == 0.1);

}


@Test

public void TestCalcResult()

{

        Rule rule = new Rule("low", "constant", "resting", "sugar");


        rule.calcResult(0.1, 0.9, 0.0, 0.35, 0.65, 0.0, 1.0, 0.0);


        assertTrue(rule.getResult() == 0.1);


        Rule rule2 = new Rule("low", null, null, "sugar");
```

```java
        rule.calcResult(0.4, 0.6, 0.0, 0.3, 0.7, 0.0, 1.0, 0.0);


        assertTrue(rule.getResult() == 0.4);

}


@Test

public void TestFindMin()

{

        Rule rule = new Rule("low", "constant", "resting", "sugar");


        assertTrue(rule.findMin(0.3, 0.55, 0.4) == 0.3);

}

}
```