Xiaobin Chen
37746611

Part 1, Step 1: Mutation analysis

1. Original:

   Live Mutants # 16
   Killed Mutants # 148
   Total Mutants # 164
   Mutant Score 90.0%

2. Added tests

   1. testTwoSegmentsDiffMoreThan500SecondLarger(),
      killed mutant is AOIU_4.

   2. testTwoSegmentsSumMoreThan500(),
      killed mutant is AORB_4

   3. testTwoSegmentsDepartureMoreThan14DaysFromNowVeryLongTime(),
      killed mutant is AORB_43

   4. testTwoSegmentsDepartureLessThan3DaysFromNowIsFreqFlierV1(),
      killed mutants are AORB_62, AORB_63, AORB_65

   5. testInvalidPrice1V1(),
      killed mutant is COR_1

   6. testInvalidPrice1Price2()
      killed mutant is COR_2

   7. testInvalidPrice1AndDepartureTime()
      killed mutant is COR_4

   8. testInvalidPrice1AndDuration()
      killed mutant is COR_6

   9. testTwoSegmentsDiffEquals500()
      killed mutant is ROR_15

10. testOneSegmentDurationEqualsTo8()
    killed mutant is ROR_22

11. testTwoSegmentsDepartureEqualsTo14DaysFromNow()
    killed mutant is ROR_29

3. Mutants that could not be killed:

    1. ROR_1.

       Because original program P first tests if price1 < 0 in

```
if (price1 < 0 || price2 < 0 || departureTime < System.currentTimeMillis() || duration < 0)
```

       and then tests if price1 > 0 in

```
if (price1 != 0 && price2 != 0)
```

       Mutant program P' tests if price1 < 0 in the same code block and then tests if price1 > 0 in

```
if (price1 > 0 && price2 != 0)  //this is the mutant code
```

       With other codes being the same, we can see that P and P' are equivalent mutants, produce the same outputs for all inputs, thus this mutant could not be killed.

    2. ROR_8

       ROR_8 is similar to ROR_1, the different is just that the affected variable is now price2 (variable price1 and price2 are equivalent). We can easily verify that the mutant program is equivalent to the original one for all inputs, using the same reason as in ROR_1, so the mutant could not be killed.

    3. ROR_37

       Because the original program P tests

```
if (price == Double.POSITIVE_INFINITY)
```

       which is equivalent to mutant program P' that tests

```
if (price >= Double.POSITIVE_INFINITY)   //because price could not be greater than Double.POSITIVE_INFINITY
```

       So P and P' are equivalent mutants, produce the same outputs for all inputs, thus this mutant could not be killed.

4. Final score:

Live Mutants #	3
Killed Mutants #	161
Total Mutants #	164
Mutant Score	98.0%

Part 1, Step 2: Mutation testing

1. Final score:

Live Mutants #	3
Killed Mutants #	86
Total Mutants #	89
Mutant Score	96.0%

2. Mutants that could not be killed:

    1. AOIU_5

    Because in the original program P, the code is:

```
    planes[i].location.x += planes[i].velocity *
Math.cos(planes[i].bearing/360.0 * 2 * Math.PI);
```

    and the mutant program P' adds an '-' sign in Math.cos(), like this:

```
   planes[i].location.x += planes[i].velocity *
Math.cos(-planes[i].bearing/360.0 * 2 * Math.PI);
```

    As we know cos(-x) == cos(x), so they are actually equivalent mutants, produce the same result for all inputs test cases.

    2. AOIU_6

    The mutant program P' adds an '-' sign in Math.cos(), like this:

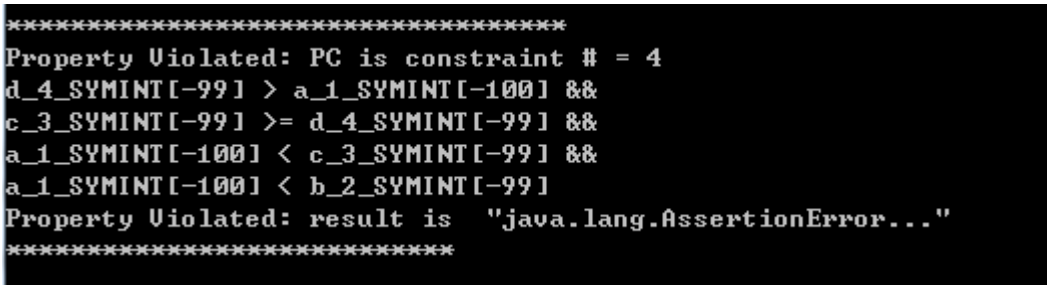
```
   planes[i].location.x += planes[i].velocity *
Math.cos(planes[i].bearing/360.0 * 2 * -Math.PI);
```

    From the analysis above for AOIU_5, we know P and P' are actually equivalent mutants, produce the same result for all inputs test cases. Thus it could not be killed.

    3. AORS_4(ignore this one as discussed in piazza)

Part 2: Model Checking
Step1

1. JPF output:



Property violated is that the result of min() is not the minimum value of inputs.
Violated assertion:

```
assert (r <= a && r <= b && r <= c && r <= d);
```

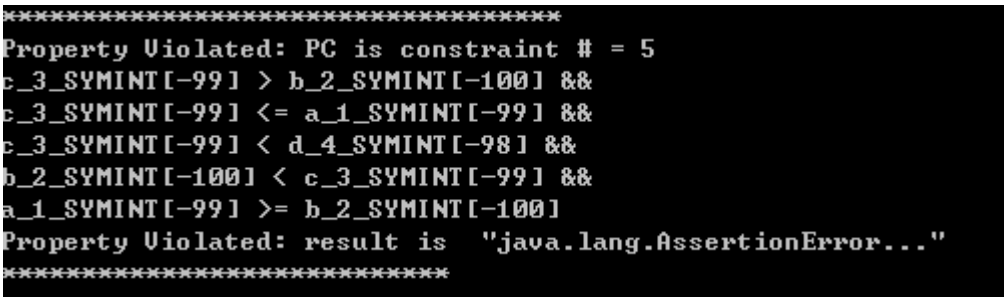
From the output, we can see that some values of *abcd* that causes violation of assertion can be that a(-100), b(-99), c(-99), d(-99).
First three lines from bottom upwards specify the path condition:

$$a < b \;\&\&\; a < c \;\&\&\; d <= c$$

we first have $a < b$, then $a < c$, then !(d>c). Then we have the violation that *d(-99)* is not less or equal to *a(-100)* as expected on this path.
So as we follow the path described above, we can find that bug. The bug is that when $a < b$ and $a < c$ and $c >= d$, the original program does not compare *a* and *d.* So add a comparison between *a* and *d* to find the minimum.

```
if (a < b) {
   if (a < c) {
      if (c < d) r = a;
      else {
         if (a < d) r = a;     //comparison between a and d
         else r = d;
      }
   }
…
}
```

2. After fixing this bug, next bug appears:



Property violated is that the result of min() is not the minimum value of inputs.
Violated assertion:

```
assert (r <= a && r <= b && r <= c && r <= d);
```

We can find the path condition is *a >= b*, then *b < c* then *c < d*:

a >= b && b < c && c < d

c(-99) satisfies *c <= a(-99),* but causes the violation because of that *c(-99)* is not less or equal to *b(-100)* as expected on this path.

The bug is that when abcd satisfy *a >= b*, then *b < c* then *c < d*, the minimum should be *b* not *c.* so change from

```
else if (b < c) {
    if (c < d) r = c;
    …
```

to

```
else if (b < c) {
    if (c < d) r = b;
    …
```

Step2

Assertion added:

```
// all elements in return array are positive
for (int i = 0; i < R.length; i++){
    assert (R[i] > 0);
}

// all elements in new array are in increasing order.
for (int i = 1; i <R.length; i++){
    assert (R[i] >= R[i - 1]);
}
```

```
            // all positive elements are in the return array R[].
            int[] original = copyArray(A);
            int index = 0;
            for (int i = 0; i < R.length; i++){
                for (index = 0; index < original.length; index++){
                    if (original[index] == R[i]){
                        original[index] = -1;      //find corresponding num in
A[] and remove it
                        break;
                    }
                }
                assert(index != original.length);     //every positive num in
R[] has corresponding positive num in A[]
            }
            for (int i = 0; i < original.length; i++){
                assert (original[i] <= 0);        //not positive num left in
A[]
            }
```

JPF output:

```
Property Violated: PC is constraint # = 14
A3_4_SYMINT[3] < A1_2_SYMINT[4] &&
A4_5_SYMINT[3] >= A2_3_SYMINT[2] &&
A2_3_SYMINT[2] >= A0_1_SYMINT[1] &&
A2_3_SYMINT[2] < A4_5_SYMINT[3] &&
A2_3_SYMINT[2] < A3_4_SYMINT[3] &&
A0_1_SYMINT[1] < A4_5_SYMINT[3] &&
A0_1_SYMINT[1] < A3_4_SYMINT[3] &&
A0_1_SYMINT[1] < A2_3_SYMINT[2] &&
A0_1_SYMINT[1] < A1_2_SYMINT[4] &&
A4_5_SYMINT[3] > CONST_0 &&
A3_4_SYMINT[3] > CONST_0 &&
A2_3_SYMINT[2] > CONST_0 &&
A1_2_SYMINT[4] > CONST_0 &&
A0_1_SYMINT[1] > CONST_0
Property Violated: result is  "java.lang.AssertionError..."
******************************
```

Property violated is that the result of sort() is not in increasing order.

Violated assertion:

```
    for (int i = 1; i <R.length; i++){
            assert (R[i] >= R[i - 1]);
    }
```

The value that caused violation is that A[] contains positive numbers {1, 4, 2, 3, 3}. After sorting, R[] is not in increasing order.

The Path condition is that:

A[i] > 0 (i = [0 ,4]) && A[0] < A[1] && A[0] < A[2] && A[0] < A[3] && A[0] < A[4] && A[2] < A[3] && A[2] < A[4]

From the path we know that the comparison of bubble sort is not correct and as a result R[0] = 4 > R[1] = 3 which violates the assertion.

The bug is that in code:

```
for (int i = 0; i < R.length; i++) {
    for (int j = i; j < R.length-1; j++) {
        if (R[j] < R[j+1]) {
            …
```

change it to

```
for (int i = 0; i < R.length; i++) {
    for (int j = 0; j < R.length-1-i; j++) {
        if (R[j] > R[j+1]) {
        …
```

After the changes, for every i, the program computes the largest value in range [0, R.length -1- i], and put it R[R.length -1 -i], so that the value in R[R.length-1-i] >= all values in range[0, R.length-1-i] in R[], which ensures the increasing order when i == R.length-1.

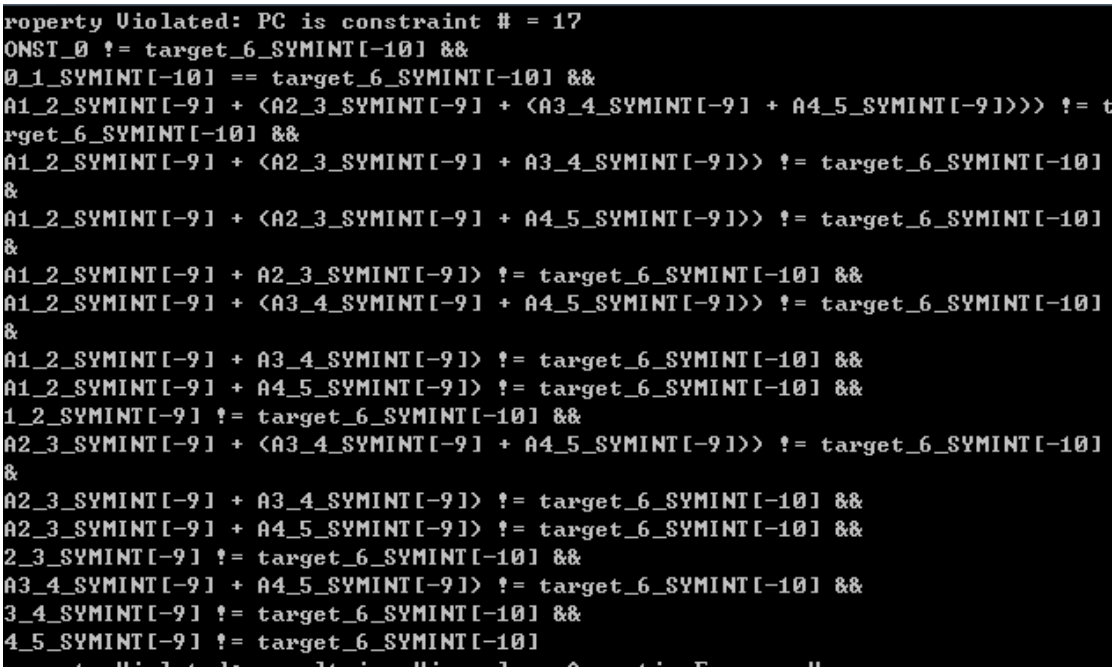
Step3

Assertion added:

```
// all the indicated return values add up to target value

if (soln != null) { //ignore null case
    int mSum = 0;
    for (int i = 0; i < A.length; i++){
        if (soln[i]){
                mSum += A[i];
        }
    }
    assert(mSum == target);
}
return soln;
```

JPF output:



Property violated is that the result of SubsetSum is not the correct value combinations that sum up to target value.
Violated assertion:

```
assert(mSum == target); //(mSum is the sum of all indicated
values that added up to the target value)
```

The input that causes the violation is that A[] = {-10, -9, -9, -9, -9}, from the path condition, we can know that the combination has been correctly found, be is not record in soln[] correctly. The bug is in code:

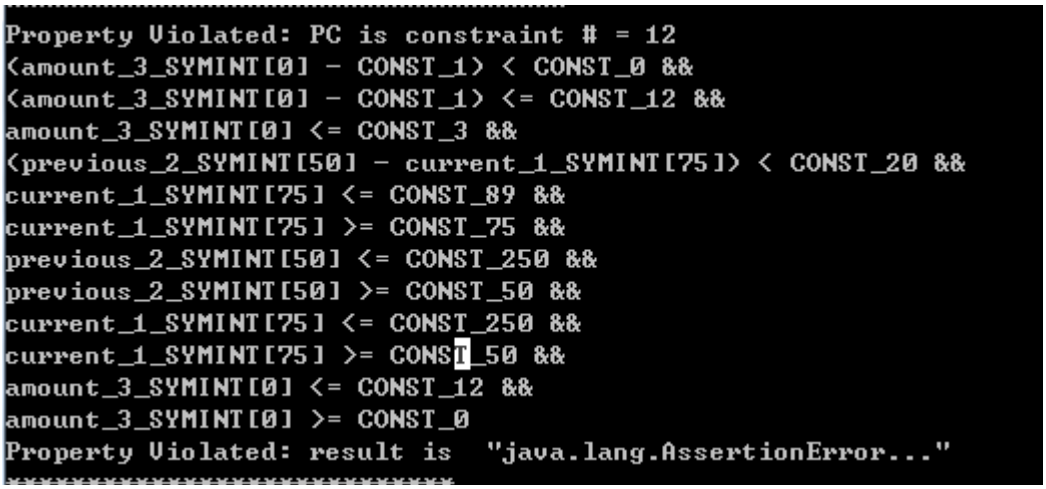
```
for (int i = A.length-1; i > 0; i--) {
```

which does not record soln[0], change it to

```
for (int i = A.length-1; i >= 0; i--) {
```

Step4

1. JPF output:



Property violated is that the result of calculate() is not between[0, 12]. Violated assertion:

```
assert(result <= 12 && result >= 0);
```

From the output, we can have the path condition:

amount >= 0 && amount <= 12 && current >= 50 && current <= 250 && previous >= 50 && previous <= 250 && current >= 75 && current <= 89 && (previous - current) < 20 && amount <= 3

When the input is amount = 0, current = 75, previous = 50, the output is -1 which violates the assertion.
So change code from

```
else if (current <= 89){
        if (decrease < 20){
            if (amount > 3){
                result = amount - 2;
            }else{
                result = amount - 1;
            }
```

To

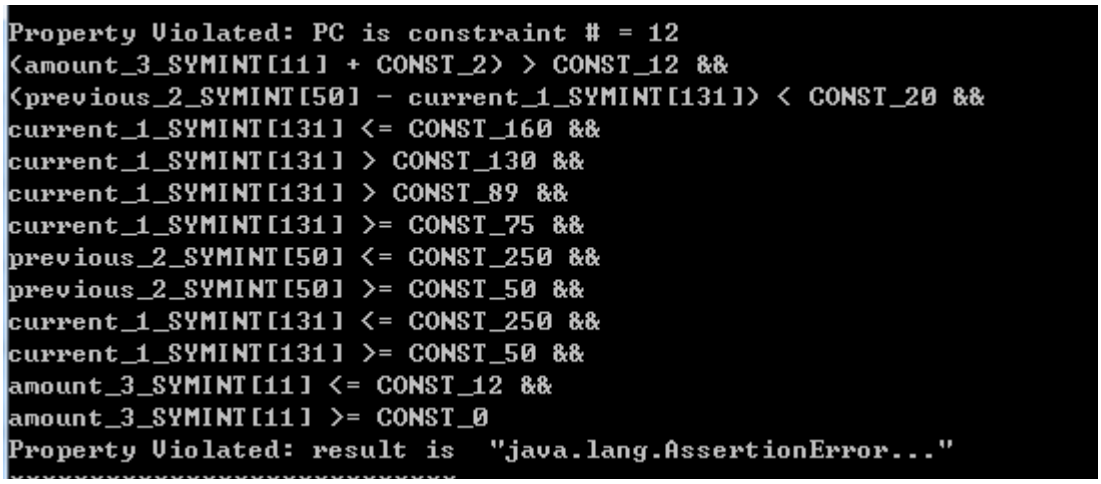
```
else if (current <= 89){
       if (decrease < 20){
           if (amount > 3){
               result = amount - 2;
           }else if (amount >= 1){
               result = amount - 1;
           }else{
```

```
                result = 0;
            }
```

New specification is that:

Range	Current BG reading	< 20	≥ 20
LOW	75 - 89	if amount > 3, reduce amount by 2 elseif amount >= 1, reduce amount by 1, else set amount = 0	set amount to 0

2. Next violation is that:



Property violated is that the result of calculate() is not between[0, 12]. Violated assertion:

```
assert(result <= 12 && result >= 0);
```

From the output, we can have the path condition:

amount >= 0 && amount <= 12 && current >= 50 && current <= 250 && previous >= 50 && previous <= 250 && current > 130 && current <= 160 && (previous - current) < 20

When the input is amount = 11, current = 131, previous = 50, the output is 13 which violates the assertion.
So change code from

```
    else if (current <= 160){
        if (decrease < 20){
            result = amount + 2;
        }
```

to

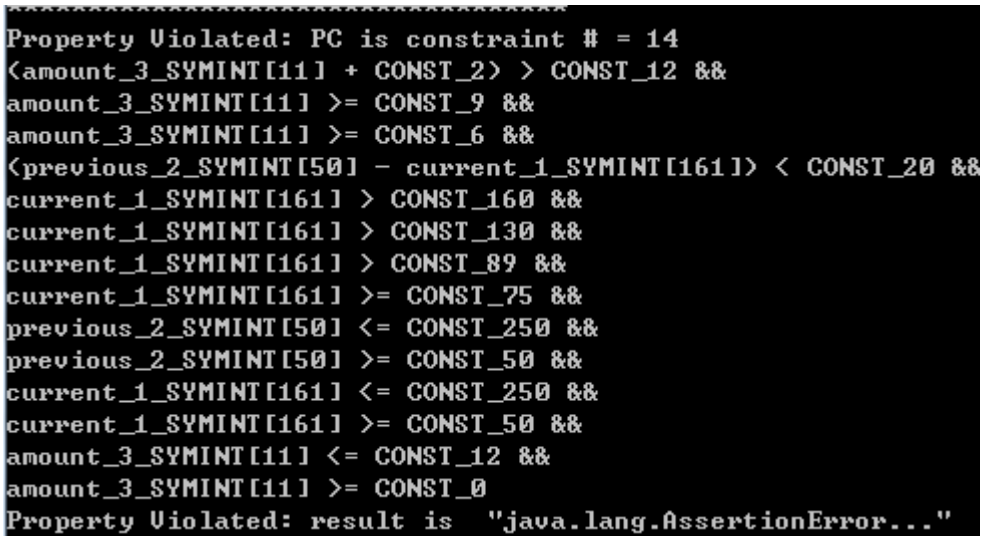
```
    else if (current <= 160){
        if (decrease < 20){
```

```
if (amount <= 10){
    result = amount + 2;
}else{
    result = 12;
}
```

New specification is that:

Range	Current BG reading	< 20	≥ 20
HIGH	131 - 160	If amount <= 10, increase amount by 2, else set amount = 12	if amount > 8, keep amount the same otherwise increase amount by 1

3. Next violation is that:



Property violated is that the result of calculate() is not between[0, 12]. Violated assertion:

```
assert(result <= 12 && result >= 0);
```

From the output, we can have the path condition:

amount >= 0 && amount <= 12 && current >= 50 && current <= 250 && previous >= 50 && previous <= 250 && current > 160 && (previous - current) < 20 && amount > 6 && amount > 9

When the input is amount = 11, current = 161, previous = 50, the output is 13 which violates the assertion.

So change code from

```
    else{
        if (decrease < 20){
            if (amount < 6){
                result = amount + 4;
            }else if (amount < 9){
                result = amount + 3;
            }else{
                result = amount + 2;
            }
```

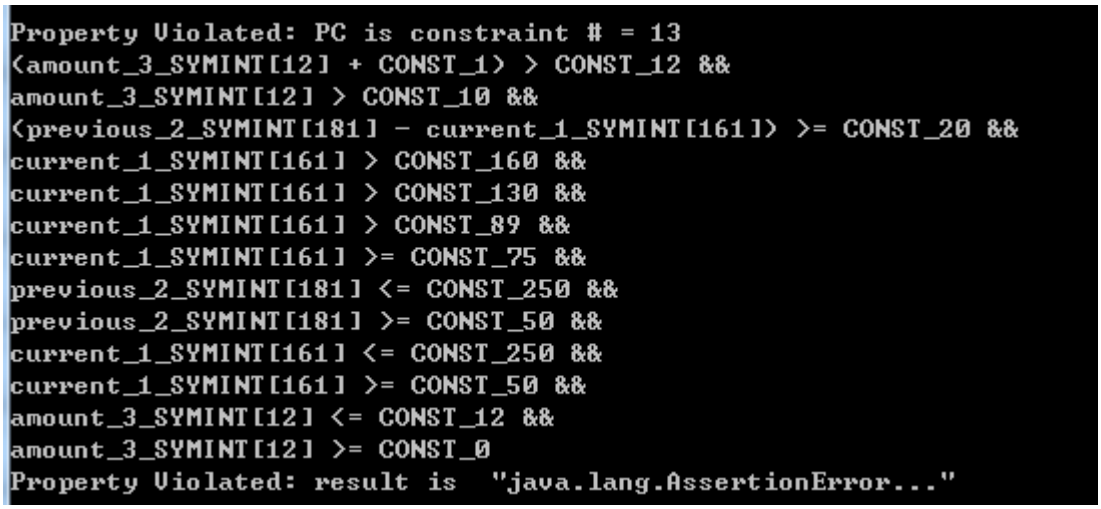
To

```
    else{
        if (decrease < 20){
            if (amount < 6){
                result = amount + 4;
            }else if (amount < 9){
                result = amount + 3;
            }else{
                if (amount <= 10){
                    result = amount + 2;
                }else{
                    result = 12;
                }
            }
```

New specification is that:

Range	**Current BG reading**	**< 20**	**≥ 20**
VERY HIGH	> 160	if amount < 6, increase amount by 4 if amount ≥ 6 and < 9, increase amount by 3 if amount <= 10 increase amount by 2 else set amount = 12	if amount ≤ 10, increase amount by 2 otherwise increase amount by 1

4. Next violation is that:



Property violated is that the result of calculate() is not between[0, 12]. Violated assertion:

```
assert(result <= 12 && result >= 0);
```

From the output, we can have the path condition:

amount >= 0 && amount <= 12 && current >= 50 && current <= 250 && previous >= 50 && previous <= 250 &&current >75 && current > 89 && current > 130 && current > 160 && (previous - current) >= 20 && amount > 10

When the input is amount = 12, current = 161, previous = 181, the output is 13 which violates the assertion.
So change code from

```
else{
    if (amount <= 10){
        result = amount + 2;
    }else{
        result = amount + 1;
    }
}
```

TO

```
else{
    if (amount <= 10){
        result = amount + 2;
    }else{
        result = 12;
    }
}
```

New specification is that:

Range	Current BG reading	< 20	≥ 20
VERY HIGH	> 160	if amount < 6, increase amount by 4 if amount ≥ 6 and < 9, increase amount by 3 if amount <= 10 increase amount by 2 else set amount = 12	if amount ≤ 10, increase amount by 2 otherwise set amount = 12

After above modifications of specification, all violations are eliminated.

The smallest value of the maximum change in BG level that can be guaranteed by this model is 48

```
assert current - newLevel <= 48;
```