Introduction

This program component provides a reusable segment of computer code to simulate the rolling of 6-sided dice. Up to 5 dice may be rolled at the discretion of the user or client software implementing this component. The output of this program component are the individual values of each die roll.

Requirements

1. The program shall simulate the rolling of up to 5 independent 6-sided dice.

    1.1 The program shall allow the user to specify the number of dice (up to a maximum of 5) to be rolled.

    1.2 The program shall allow the client software to specify the number of dice (up to a maximum of 5) to be rolled.

    1.3 The program shall generate random integers for an individual die roll based on a discrete uniform probability distribution.

2. The program shall provide the individual value of each die roll as an obtainable output.

3. The program shall be structured in an encapsulated manner that supports reuse across multiple computer programs.

Jeffrey M. Borkowski
9 Sep 2015
CEG-3110
Project 1

Test Plan

This test plan verifies the ability of the program component to operate in satisfaction of the requirements listed above. During this phase of testing, tests of user input are omitted. Instead, tests focus on demonstrating the nature of the simulated dice rolls and providing evidence of the randomness of the rolls. The individual dice roll results are expected to occur with a uniform probability of approximately 6^{-n}, where n is the number of dice rolled at a time using a given instance of the program component. Additionally, the probability distribution of the sum of the n dice is expected to approach that of a discrete normal distribution as n increases. Given the nature of empirical probability values, tests designed to verify the underlying probability distribution used in this program component will compare experimental values against a range of acceptable values.

Test Case #1

This test case will verify the ability of the program component to roll 1 die at a time. A total of ten rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 1 die at a time.

Required input: N/A

Expected output:

Test Case #1
(*), (*), (*), (*), (*), (*), (*), (*), (*), (*)

Note to tester: Each * in the above expected output may be any particular number from the set {1, 2, 3, 4, 5, 6}.

Actual output:

Test Case #1
(6), (4), (3), (3), (5), (6), (3), (3), (5), (5)

Result: Pass ✓ Fail _____

Test Case #2

This test case will verify the ability of the program component to roll 2 dice at a time. A total of ten pairs of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 2 dice at a time.

Required input: N/A

Expected output:

Test Case #2
(*, *), (*, *), (*, *), (*, *), (*, *), (*, *), (*, *), (*, *), (*, *), (*, *)

Note to tester: Each * in the above expected output may be any particular number from the set {1, 2, 3, 4, 5, 6}.

Actual output:

Test Case #2
(2, 3), (3, 5), (4, 2), (1, 4), (3, 2), (4, 4), (6, 5), (3, 1), (6, 3), (4, 3)

Result: Pass ✓ Fail _____

Test Case #3

This test case will verify the ability of the program component to roll 3 dice at a time. A total of ten triples of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 3 dice at a time.

Required input: N/A

Expected output:

Test Case #3
(*, *, *), (*, *, *), (*, *, *), (*, *, *), (*, *, *), (*, *, *), (*, *, *), (*, *, *), (*, *, *), (*, *, *)

Note to tester: Each * in the above expected output may be any particular number from the set {1, 2, 3, 4, 5, 6}.

Actual output:

Test Case #3
(2, 4, 5), (2, 3, 3), (3, 2, 6), (3, 4, 4), (3, 2, 4), (5, 1, 2), (1, 5, 3), (3, 4, 6), (1, 3, 2), (4, 5, 1)

Result: Pass ✓ Fail _____

Test Case #4

This test case will verify the ability of the program component to roll 4 dice at a time. A total of ten quadruples of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 4 dice at a time.

Required input: N/A

Expected output:

Test Case #4
(*, *, *, *), (*, *, *, *), (*, *, *, *), (*, *, *, *), (*, *, *, *), (*, *, *, *), (*, *, *, *), (*, *, *, *), (*, *, *, *), (*, *, *, *)

Note to tester: Each * in the above expected output may be any particular number from the set {1, 2, 3, 4, 5, 6}.

Actual output:

Test Case #4
(2,4,6,2), (6,1,2,2), (6,2,4,4), (6,5,1,3), (6,6,4,1), (2,4,4,6), (1,1,3,1), (1,5,3,5), (2,2,6,4), (4,1,3,2)

Result: Pass ✓ Fail _____

Test Case #5

This test case will verify the ability of the program component to roll 5 dice at a time. A total of ten quintuples of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 5 dice at a time.

Required input: N/A

Expected output:

Test Case #5
(*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *), (*, *, *, *, *)

Note to tester: Each * in the above expected output may be any particular number from the set {1, 2, 3, 4, 5, 6}.

Actual output:

Test Case #5
(2,4,6,1,3), (5,2,3,2,1), (5,1,4,1,6), (5,5,3,3,6), (2,3,6,6,6), (2,1,2,1,1), (1,1,6,6,2), (6,5,1,3,5), (4,5,4,6,5), (5,2,1,2,2)

Result: Pass ✓ Fail _____

Test Case #6

This test case will verify that rolls of 1 die at a time appear to be random, with an approximately uniform probability distribution. A total of at least 1 million rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 1 die at a time.

Required input: N/A

Expected output:

Test Case #6

1: (0.1600 < x < 0.1750) 2: (0.1600 < x < 0.1750)
3: (0.1600 < x < 0.1750) 4: (0.1600 < x < 0.1750)
5: (0.1600 < x < 0.1750) 6: (0.1600 < x < 0.1750)
Automatically detected errors: 0

Note to tester: The inequalities listed in parentheses in the expected output signify the acceptable range of values for each segment of the output. So long as each output value is simultaneously greater than its respective lower bound and less than its respective upper bound, this portion of the test may be considered successful.

Actual output:

Test Case #6
1: 0.166525 2: 0.167314
3: 0.167069 4: 0.166593
5: 0.165886 6: 0.166613
Automatically detected errors: 0

Result: Pass ✓ Fail _____

Test Case #7

This test case will verify that rolls of 2 dice at a time appear to be random, with an approximately uniform probability distribution of each 2-tuple of die values. A total of at least 1 million pairs of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 2 dice at a time.

Required input: N/A

Expected output:

Test Case #7
1: 0 2: (0.0225 < x < 0.0325)
3: (0.0500 < x < 0.0605) 4: (0.0783 < x < 0.0883)
5: (0.1061 < x < 0.1161) 6: (0.1338 < x < 0.1438)
7: (0.1616 < x < 0.1716) 8: (0.1338 < x < 0.1438)
9: (0.1061 < x < 0.1161) 10: (0.0783 < x < 0.0883)
11: (0.0500 < x < 0.0605) 12: (0.0225 < x < 0.0325)
Automatically detected errors: 0

Note to tester: The inequalities listed in parentheses in the expected output signify the acceptable range of values for each segment of the output. So long as each output value is simultaneously greater than its respective lower bound and less than its respective upper bound, this portion of the test may be considered successful.

Actual output:

Test Case #7

1: 0 2: 0.027942

3: 0.055697 4: 0.083279

5: 0.111081 6: 0.138718

7: 0.166721 8: 0.138383

9: 0.111285 10: 0.083325

11: 0.055934 12: 0.027635

Automatically detected errors: 0

Result: Pass ✓ Fail _____

Test Case #8

This test case will verify that rolls of 3 dice at a time appear to be random, with an approximately uniform probability distribution of each 3-tuple of die values. A total of at least 1 million triples of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 3 dice at a time.

Required input: N/A

Expected output:

Test Case #8
1: 0 2: 0
3: (0.004129 < x < 0.005129) 4: (0.008888 < x < 0.01888)
5: (0.02277 < x < 0.03277) 6: (0.04129 < x < 0.05129)
7: (0.06444 < x < 0.07444) 8: (0.09222 < x < 0.1022)
9: (0.1107 < x < 0.1207) 10: (0.1200 < x < 0.1300)
11: (0.1200 < x < 0.1300) 12: (0.1107 < x < 0.1207)
13: (0.09222 < x < 0.1022) 14: (0.06444 < x < 0.07444)
15: (0.04129 < x < 0.05129) 16: (0.02277 < x < 0.03277)
17: (0.008888 < x < 0.01888) 18: (0.004129 < x < 0.005129)
Automatically detected errors: 0

Note to tester: The inequalities listed in parentheses in the expected output signify the acceptable range of values for each segment of the output. So long as each output value is simultaneously greater than its respective lower bound and less than its respective upper bound, this portion of the test may be considered successful.

Actual output:

Test Case #8

1: 0 2: 0

3: 0.004592 4: 0.01407
5: 0.02763 6: 0.045932
7: 0.069481 8: 0.09679
9: 0.116181 10: 0.125401
11: 0.125189 12: 0.115567
13: 0.097167 14: 0.069398
15: 0.046439 16: 0.027847
17: 0.013719 18: 0.004597
Automatically detected errors: 0

Result: Pass ✓ Fail _____

Test Case #9

This test case will verify that rolls of 4 dice at a time appear to be random, with an approximately uniform probability distribution of each 4-tuple of die values. A total of at least 1 million quadruples of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 4 dice at a time.

Required input: N/A

Expected output:

Test Case #9
1: 0 2: 0
3: 0.~~0000~~ 4: (0.0007216 < x < 0.0008216)
5: (0.002586 < x < 0.003586) 6: (0.007216 < x < 0.008216)
7: (0.01043 < x < 0.02043) 8: (0.02200 < x < 0.03200)
9: (0.03820 < x < 0.04820) 10: (0.05672 < x < 0.06672)
11: (0.07524 < x < 0.08524) 12: (0.09145 < x < 0.1014)
13: (0.1030 < x < 0.1130) 14: (0.1076 < x < 0.1176)
15: (0.1030 < x < 0.1130) 16: (0.09145 < x < 0.1014)
17: (0.07524 < x < 0.08524) 18: (0.05672 < x < 0.06672)
19: (0.03820 < x < 0.04820) 20: (0.02200 < x < 0.03200)
21: (0.01043 < x < 0.02043) 22: (0.007216 < x < 0.008216)
23: (0.002586 < x < 0.003586) 24: (0.0007216 < x < 0.0008216)
Automatically detected errors: 0

Note to tester: The inequalities listed in parentheses in the expected output signify the acceptable range of values for each segment of the output. So long as each output value is simultaneously greater than its respective lower bound and less than its respective upper bound, this portion of the test may be considered successful.

Actual output:

Test Case #9
1:0 2:0
3:0 4:0.000715
5:0.00307 6:0.007625
7:0.015379 8:0.027313
9:0.043015 10:0.061689
11:0.080029 12:0.096261
13:0.108433 14:0.112941
15:0.108173 16:0.096786
17:0.079881 18:0.061874
19:0.042909 20:0.026864
21:0.015436 22:0.007747
23:0.00313 24:0.00073
Automatically detected errors: 0

Result: Pass _____ Fail ✓

Test Case #10

This test case will verify that rolls of 5 dice at a time appear to be random, with an approximately uniform probability distribution of each 5-tuple of die values. A total of at least 5 million quintuples of rolls will be simulated.

Initial conditions: A new dice roller object is available and configured to roll 5 dice at a time.

Required input: N/A

Expected output:

Test Case #10
1: 0 2: 0
3: 0 4: 0
5: (0.0000786 < x < 0.0001786) 6: (0.0005930 < x < 0.0006930)
7: (0.001429 < x < 0.002429) 8: (0.004001 < x < 0.005001)
9: (0.008502 < x < 0.009502) 10: (0.01120 < x < 0.02120)
11: (0.02136 < x < 0.03136) 12: (0.03422 < x < 0.04422)
13: (0.04901 < x < 0.05901) 14: (0.06444 < x < 0.07444)
15: (0.07871 < x < 0.08871) 16: (0.08952 < x < 0.09952)
17: (0.0953 < x < 0.1053) 18: (0.0953 < x < 0.1053)

19: (0.08952 < x < 0.09952) 20: (0.07871 < x < 0.08871)
21: (0.06444 < x < 0.07444) 22: (0.04901 < x < 0.05901)
23: (0.03422 < x < 0.04422) 24: (0.02136 < x < 0.03136)
25: (0.01120 < x < 0.02120) 26: (0.008502 < x < 0.009502)
27: (0.004001 < x < 0.005001) 28: (0.001429 < x < 0.002429)
29: (0.0005930 < x < 0.0006930) 30: (0.0000786 < x < 0.0001786)
Automatically detected errors: 0

Note to tester: The inequalities listed in parentheses in the expected output signify the acceptable range of values for each segment of the output. So long as each output value is simultaneously greater than its respective lower bound and less than its respective upper bound, this portion of the test may be considered successful.

Actual output:

Test Case #10
1:0 2:0
3:0 4:0
5:0.0001292 6:0.0006274
7:0.0019214 8:0.0045156
9:0.0089442 10:0.01636
11:0.0262516 12:0.0392506
13:0.0541394 14:0.0693368
15:0.0835416 16:0.0946942
17:0.100236 18:0.100455
19:0.0943784 20:0.0837826
21:0.0694058 22:0.053963
23:0.0391856 24:0.0264258
25:0.016272 26:0.0089294
27:0.0045304 28:0.001946
29:0.0006476 30:0.0001308
Automatically detected errors: 0

Result: Pass ✓ Fail _____

Summary

I began my solution to this project by viewing the introductory section of the project assignment as the result of an independent analysis of the user's needs. Next, I translated these user needs

into the requirements for a software component. These requirements intentionally specified only **what** needed to be designed and built, while leaving the **how** aspect to the discretion of the programmer.

After writing these requirements, I drafted the test plan and specified test cases to investigate each possible value of user input (selection of number of dice to roll), while observing the probabilities of output events. For the simple case of rolling a single die at a time, it was sufficient to verify that the output followed an approximately uniform discrete probability distribution (with $p = 1/6$). As the concurrent number of dice rolled increased, it was necessary to verify the probability distributions of both the sum of the dice and the roll outcomes (viewed as a tuple of individual die rolls). The latter distribution was expected to remain approximately uniform (with $p = 6^{-n}$, where n is the concurrent number of dice rolled), while it was anticipated that the former distribution would approach that of a discrete normal distribution as the concurrent number of dice rolled increased. In order to generate a sufficiently large set of data to compute the observed probabilities, it was necessary to simulate at least 1 million dice roll events. For the case of rolling 5 dice at a time, the number of dice roll events had to be increased to at least 5 million due to the significantly larger number of possible outcomes.

Based on the successful completion of the test plan above, I feel confident in stating that this program component simulates the rolling of dice in a sufficiently random manner.

* Note: The failure noted in test case #9 may be retroactively accepted because it only busted the bound requirements on a single segment. This occurs as a result of computing empirical probability values. Successive runs of the test program would typically <u>not</u> demonstrate this failure type.

Jeffrey M. Borkowski
9 Sep 2015
CEG-3110
Project 1

Sample Output of Test Script

Test Case #1
(6), (4), (3), (3), (5), (6), (3), (3), (5), (5)

Test Case #2
(2, 3), (3, 5), (4, 2), (1, 4), (3, 2), (4, 4), (6, 5), (3, 1), (6, 3), (4, 3)

Test Case #3
(2, 4, 5), (2, 3, 3), (3, 2, 6), (3, 4, 4), (3, 2, 4), (5, 1, 2), (1, 5, 3), (3, 4, 6), (1, 3, 2), (4, 5, 1)

Test Case #4
(2, 4, 6, 2), (6, 1, 2, 2), (6, 2, 4, 4), (6, 5, 1, 3), (6, 6, 4, 1), (2, 4, 4, 6), (1, 1, 3, 1), (1, 5, 3, 5), (2, 2, 6, 4), (4, 1, 3, 2)

Test Case #5
(2, 4, 6, 1, 3), (5, 2, 3, 2, 1), (5, 1, 4, 1, 6), (5, 5, 3, 3, 6), (2, 3, 6, 6, 6), (2, 1, 2, 1, 1), (1, 1, 6, 6, 2), (6, 5, 1, 3, 5), (4, 5, 4, 6, 5), (5, 2, 1, 2, 2)

Test Case #6
1: 0.166525 2: 0.167314
3: 0.167069 4: 0.166593
5: 0.165886 6: 0.166613
Automatically detected errors: 0

Test Case #7
1: 0 2: 0.027942
3: 0.055697 4: 0.083279
5: 0.111081 6: 0.138718
7: 0.166721 8: 0.138383
9: 0.111285 10: 0.083325
11: 0.055934 12: 0.027635
Automatically detected errors: 0

Test Case #8
1: 0 2: 0
3: 0.004592 4: 0.01407
5: 0.02763 6: 0.045932
7: 0.069481 8: 0.09679
9: 0.116181 10: 0.125401
11: 0.125189 12: 0.115567
13: 0.097167 14: 0.069398
15: 0.046439 16: 0.027847
17: 0.013719 18: 0.004597
Automatically detected errors: 0

Test Case #9
1: 0 2: 0
3: 0 4: 0.000715
5: 0.00307 6: 0.007625
7: 0.015379 8: 0.027313
9: 0.043015 10: 0.061689
11: 0.080029 12: 0.096261
13: 0.108433 14: 0.112941
15: 0.108173 16: 0.096786
17: 0.079881 18: 0.061874
19: 0.042909 20: 0.026864
21: 0.015436 22: 0.007747
23: 0.00313 24: 0.00073
Automatically detected errors: 0

Test Case #10
1: 0 2: 0
3: 0 4: 0
5: 0.0001292 6: 0.0006274
7: 0.0019214 8: 0.0045156
9: 0.0089442 10: 0.01636
11: 0.0262516 12: 0.0392506
13: 0.0541394 14: 0.0693368
15: 0.0835416 16: 0.0946942
17: 0.100236 18: 0.100455
19: 0.0943784 20: 0.0837826
21: 0.0694058 22: 0.053963
23: 0.0391856 24: 0.0264258
25: 0.016272 26: 0.0089294
27: 0.0045304 28: 0.001946
29: 0.0006476 30: 0.0001308
Automatically detected errors: 0

```
///
/// CEG-3310 Project 1 - Tester file
///
/// Created by Jeffrey M. Borkowski on 6 Sep 2015
///
#include <iostream>
#include <cmath>
#include "DieRoller.h"

void testSetA(int numDice, int numRolls)
{
   DieRoller testRoller(numDice);

   for (int i = 0; i < numRolls; ++i)
   {
      testRoller.rollDice();
      std::cout << "(";
      for (int j = 0; j < numDice - 1; ++j)
         std::cout << testRoller.getCurrentDieValue(j) << ", ";
      std::cout << testRoller.getCurrentDieValue(numDice - 1) << ")";
      if (i < numRolls - 1)
         std::cout << ", ";
      else
         std::cout << std::endl;
   }
}

void testSetB(int numDice, int numRolls)
{
   DieRoller testRoller(numDice);
   int errorCount = 0;
   double errorThreshold = 1 / (std::pow(6, numDice));

   int sumCount[6 * numDice];
   for (int i = 0; i < (6 * numDice); ++i)
      sumCount[i] = 0;

   int tupleCount[(int)(std::pow(6, numDice))];
   for (int i = 0; i < (int)(std::pow(6, numDice)); ++i)
      tupleCount[i] = 0;

   for (int i = 0; i < numRolls; ++i)
   {
      testRoller.rollDice();
      int sum = 0;
      int tupleIndex = 0;
      for (int j = 0; j < numDice; ++j)
      {
         sum += testRoller.getCurrentDieValue(j);
         tupleIndex += (int)(std::pow(6, j)) *
            (testRoller.getCurrentDieValue(j) - 1);
      }
```

```
      sumCount[sum - 1] += 1;
      tupleCount[tupleIndex] += 1;
   }

   for (int i = 0; i < (6 * numDice); ++i)
   {
      std::cout << (i + 1) << ": " << (sumCount[i] / (double)numRolls);
      if (i % 2 == 0)
         std::cout << "      ";
      else
         std::cout << std::endl;
   }

   for (int i = 0; i < (int)(std::pow(6, numDice)); ++i)
   {
      double testValue = tupleCount[i] / (double)numRolls;
      if (testValue <= (0.8 * errorThreshold) ||
          testValue >= (1.2 * errorThreshold))
         errorCount++;
   }
   std::cout << "Automatically detected errors: " << errorCount << std::endl;
}

int main(int argc, const char * argv[])
{
   DieRoller oneDie(1);
   DieRoller twoDice(2);

   std::cout << "Test Case #1" << std::endl;
   testSetA(1, 10);
   std::cout << std::endl;

   std::cout << "Test Case #2" << std::endl;
   testSetA(2, 10);
   std::cout << std::endl;

   std::cout << "Test Case #3" << std::endl;
   testSetA(3, 10);
   std::cout << std::endl;

   std::cout << "Test Case #4" << std::endl;
   testSetA(4, 10);
   std::cout << std::endl;

   std::cout << "Test Case #5" << std::endl;
   testSetA(5, 10);
   std::cout << std::endl;

   std::cout << "Test Case #6" << std::endl;
   testSetB(1, 1000000);
   std::cout << std::endl;
```

```
    std::cout << "Test Case #7" << std::endl;
    testSetB(2, 1000000);
    std::cout << std::endl;

    std::cout << "Test Case #8" << std::endl;
    testSetB(3, 1000000);
    std::cout << std::endl;

    std::cout << "Test Case #9" << std::endl;
    testSetB(4, 1000000);
    std::cout << std::endl;

    std::cout << "Test Case #10" << std::endl;
    testSetB(5, 5000000);
    std::cout << std::endl;

    return 0;
}
```

```
///
/// CEG-3310 Project 1 - DieRoller class
///
/// Created by Jeffrey M. Borkowski on 6 Sep 2015
///
#include "DieRoller.h"

DieRoller::DieRoller(int numDice)
{
   this->numDice = numDice;
   generators = std::vector<std::default_random_engine>(numDice);
   currentDiceValues = std::vector<int>(numDice);
   for (int i = 0; i < numDice; ++i)
   {
      unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
      generators[i] = std::default_random_engine (seed * (i + 1));
   }
   distribution = std::uniform_int_distribution<int> (1,6);
}

void DieRoller::rollDice()
{
   for (int i = 0; i < numDice; ++i)
   {
      currentDiceValues[i] = distribution(generators[i]);
   }
}

int DieRoller::getCurrentDieValue(int dieIndex)
{
   return currentDiceValues[dieIndex];
}
```

```
///
/// CEG-3310 Project 1 - DieRoller header file
///
/// Created by Jeffrey M. Borkowski on 6 Sep 2015
///
#ifndef __Project1__DieRoller
#define __Project1__DieRoller

#include <chrono>
#include <random>
#include <vector>

class DieRoller
{
    std::vector<std::default_random_engine> generators;
    std::uniform_int_distribution<int> distribution;
    int numDice;
    std::vector<int> currentDiceValues;

public:
    DieRoller(int numDice);

    ~DieRoller(){}

    void rollDice();
    int getCurrentDieValue(int dieNumber);
};

#endif /* defined(__Project1__DieRoller) */
```