RandomNumberGeneration – Walkthrough

Random Number Generation

If we want to develop casino style game-playing programs then we need to be able to implement an element of chance…

The element of chance can be introduced into computer applications with the Random class (located in namespace System)

Consider the following statements:

**Random randomObject = new Random();**

**int randomNumber = randomObject.Next();**

The Next method generates a positive int value between zero and the constant Int32.MaxValue (the value 2,147,483,647)

If Next produces values at random, every value in this range has an equal chance (or probability) of being chosen when Next is called

Note that values returned by Next are actually pseudo-random numbers - a sequence of values produced by a complex mathematical calculation

A seed value is required in this mathematical calculation

When we create our Random object, we use the current time of day as the seed

A particular seed value always produces the same series of random numbers

Programmers commonly use the current time of day as a seed value, since it changes each second and, therefore produces different random-number sequences each time the program executes

The range of values produced directly by Next (remember it ranges from 0 to 2,147,483,647) often is different from the range of values required in a particular application

For example, a program that simulates coin-tossing might require only 0 for “heads” and 1 for “tails”

A program that simulates rolling a six-sided die would require random integers in the range 1 – 6

A video-game program that randomly predicts the next type of spaceship (out of four possibilities) that will fly across the horizon might require random integers in the range 1 - 4

The one-argument version of method Next returns values in the range from 0 up to (but not including) the value of that argument

For example,

**value = randomObject.Next( 6 );**

produces values from 0 through 5

This is called scaling, because the range of values produced has been scaled down from over two billion to only six

The number 6 is the scaling factor

The two-argument version of method Next allows us to shift and scale the range of numbers

For example, we can use method Next as follows

**value = randomObject.Next( 1, 7 );**

to produce integers in the range from 1 to 6

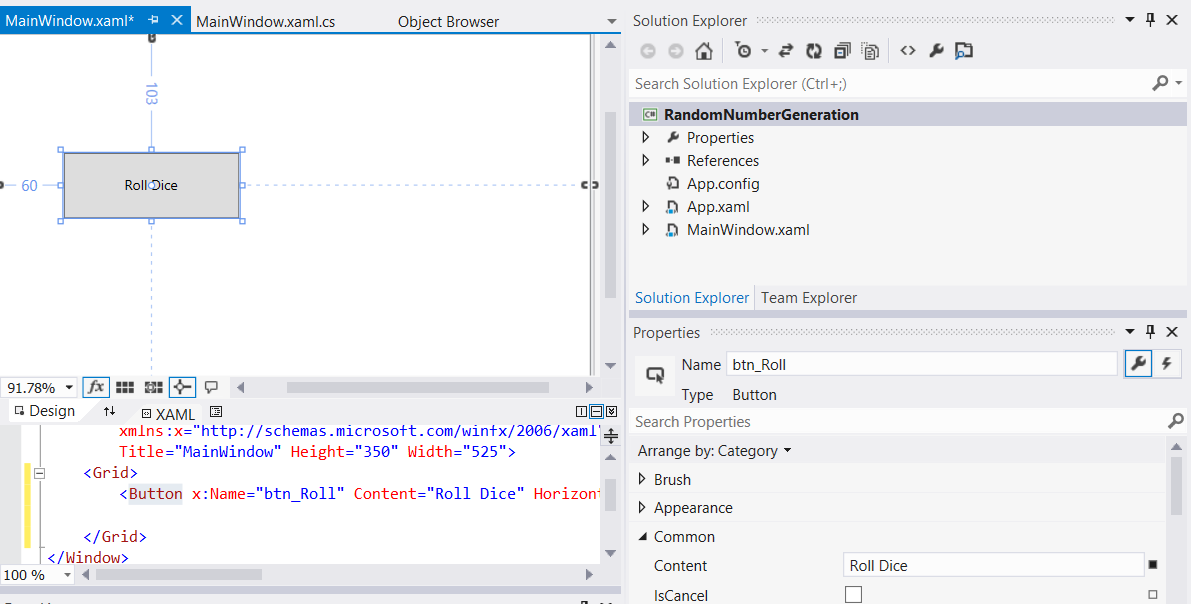
In this case, we have shifted the numbers to produce a range from 1 up to (but not including) 7

We are going to build a basic application which has a single button, which when clicked will:

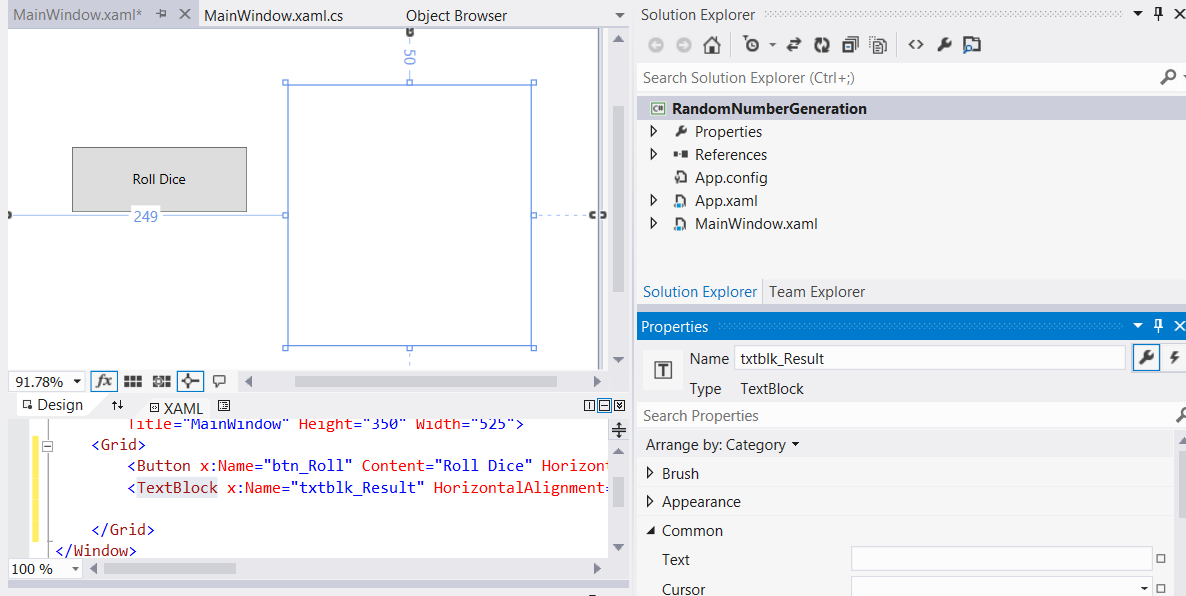
* simulates 20 rolls of a six-sided die and shows the integer value of each roll:

The aim of this tutorial is to show how to use random number generation, which can be useful in many different ways within an application.

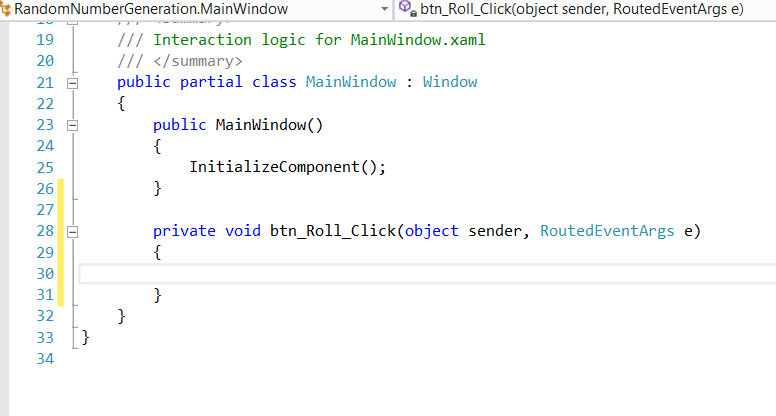
1. Create a new WPF project
2. Drag across a button from the toolbox
   1. set the “Name” property to btn\_Roll
   2. set the “Content” property to “Roll dice”



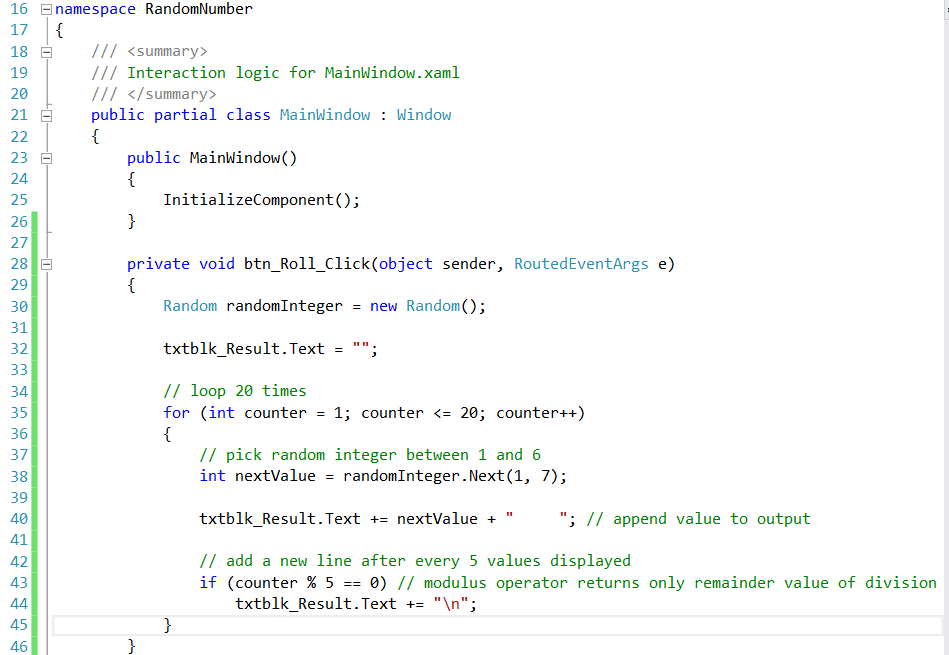
1. Drag across a text block and from the toolbox
   1. set the “Name” property to txtblk\_Result
   2. set the “Text” property to blank

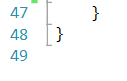


1. Double-click on the “Roll Dice” button to automatically create an event handler in the MainWindow.xaml.cs file, which is a method called btn\_Roll\_Click



1. We are now going to write the code for this event handler. The final code can be viewed in the screenshot below followed by a full explanation





Random integers in the range 1 to 6 code walkthrough

The dice-rolling simulation begins when the user clicks the Roll Dice button, which invokes the btn\_Roll\_Click event handler (lines 28 - 46)

The for loop on lines 35 - 45 repeatedly invokes method Next of class Random to simulate rolling the die

Line 40 appends the value rolled to txtblk\_Result Text property

After every five rolls, line 44 appends a newline character to make the output more readable