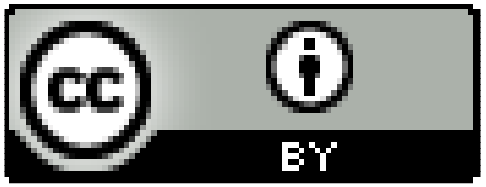
Program Design

Invasion Percolation: Randomness



5	3	7	2	6	1	1	3	4
8	5	6	5	7	2	3	6	2
2	5	8	7	5	5	6	5	9
5	2	6	4	9	3	9	6	5
4	6	8	8	5	9	7	3	9
7	6	4	5	1	2	6	8	5
5	4	2	5	8	5	5	5	8
5	7	5	1	5	3	8	5	5
4	5	1	9	7	8	6	5	1

Need a 2D grid of random values

5	3	7	2	6	1	1	3	4
8	5	6	5	7	2	3	6	2
2	5	8	7	5	5	6	5	9
5	2	6	4	9	3	9	6	5
4	6	8	8	5	9	7	3	9
7	6	4	5	1	2	6	8	5
5	4	2	5	8	5	5	5	8
5	7	5	1	5	3	8	5	5
4	5	1	9	7	8	6	5	1

Need a 2D grid of random values

Uniformly distributed in some range 1..Z

5	3	7	2	6	1	1	3	4
8	5	6	5	7	2	3	6	2
2	5	8	7	5	5	6	5	9
5	2	6	4	9	3	9	6	5
4	6	8	8	5	9	7	3	9
7	6	4	5	1	2	6	8	5
5	4	2	5	8	5	5	5	8
5	7	5	1	5	3	8	5	5
4	5	1	9	7	8	6	5	1

Need a 2D grid of random values

Uniformly distributed in some range 1..Z

Need to check the science on that...

```
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
grid = []
for x in range(N):
  grid.append([])
  for y in range(N):
    grid[-1].append(1)
```

```
from random import seed, randint
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
assert Z > 0, "Range must be positive“
seed(S)
grid = []
for x in range(N):
  grid.append([])
  for y in range(N):
    grid[-1].append(randint(1, Z))
```

```
from random import seed, randint
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
assert Z > 0, "Range must be positive“
seed(S)
grid = []
for x in range(N):
  grid.append([])
  for y in range(N):
    grid[-1].append(randint(1, Z))
```

```
>>> from random import seed, randint
>>> seed(4713983)
>>> for i in range(5):
...   print randint(1, 10),
...
7 2 6 6 5
```

```
>>> from random import seed, randint
>>> seed(4713983)
>>> for i in range(5):
...     print randint(1, 10),
...
7 2 6 6 5
```

Standard Python random number library

```
>>> from random import seed, randint
>>> seed(4713983)
>>> for i in range(5):
...     print randint(1, 10),
...
7 2 6 6 5
```

Initialize the sequence of "random" numbers

```
>>> from random import seed, randint
>>> seed(4713983)
>>> for i in range(5):
...     print randint(1, 10),
...
7 2 6 6 5
```

Produce the next "random" number in the sequence

Here's a simple "random" number generator:

```
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...   value = (3 * value + 5) % base
...   print value,
...
```

0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14

```
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
```

0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14

base controls how long before values repeat

```
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
```

0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14

base controls how long before values repeat

Once they do, values appear in exactly the same order as before

```
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
```

0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14

The *seed* controls where the sequence starts

```
>>> base = 17  # a prime
>>> value = 9  # anything in 0..base-1
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
```

15 16 2 11 4 0 5 3 14 13 10 1 8 12 7 9 15 16 2 11

The *seed* controls where the sequence starts

Changing the seed slides values left or right

```
>>> base = 17  # a prime
>>> value = 9  # anything in 0..base-1
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
```

15 16 2 11 4 0 5 3 14 13 10 1 8 12 7 9 15 16 2 11

The *seed* controls where the sequence starts

Changing the seed slides values left or right

(We'll use this fact when testing our program)

This is a lousy “random” number generator

This is a lousy “random” number generator

Did you notice that 6 never appeared?

This is a lousy “random” number generator

Did you notice that 6 never appeared?

That would probably distort our results...

What happens when 6 *does* appear?

```
>>> base = 17
>>> value = 6
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```

What happens when 6 *does* appear?

```
>>> base = 17
>>> value = 6
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
```

6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6

How can we prove this won't ever happen?

What happens when 6 *does* appear?

```
>>> base = 17
>>> value = 6
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
```

6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6

How can we prove this won't ever happen?

Or that something subtler won't go wrong?

Computers can't generate real random numbers

Computers can't generate real random numbers

But they *can* generate numbers with many of the same statistical properties as the real thing

Computers can't generate real random numbers

But they *can* generate numbers with many of the same statistical properties as the real thing

This is very hard to get right

Computers can't generate real random numbers

But they *can* generate numbers with many of the same statistical properties as the real thing

This is very hard to get right

Never try to do it yourself

Computers can't generate real random numbers

But they *can* generate numbers with many of the same statistical properties as the real thing

This is very hard to get right

Never try to do it yourself

Always use a good library

Computers can't generate real random numbers

But they *can* generate numbers with many of the same statistical properties as the real thing

This is very hard to get right

Never try to do it yourself

Always use a good library

...like Python's

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number. There are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.

– John von Neumann

created by

Greg Wilson

May 2010