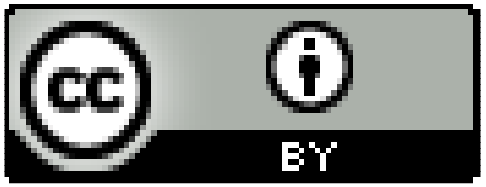
Program Design

Invasion Percolation: Testing



We found one bug

We found one bug

How many others *haven't* we found?

We found one bug

How many others *haven't* we found?

How do we *validate* and *verify* this program?

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

- Validation: are we using a good model?

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

- Validation: are we using a good model?

The second question is a question for scientists...

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

- Validation: are we using a good model?

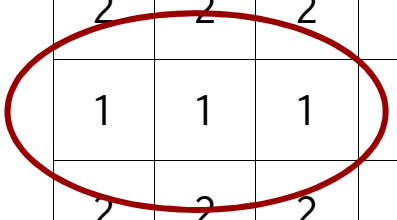
The second question is a question for scientists...

...so we'll concentrate on testing our program

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2



This grid...

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
−1	−1	−1	2	2
2	2	2	2	2
2	2	2	2	2

...should fill in like this

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
-1	-1	-1	2	2
2	2	2	2	2
2	2	2	2	2

...should fill in like this

If it doesn't, it should be easy to figure out why not

Reminder of revised program structure

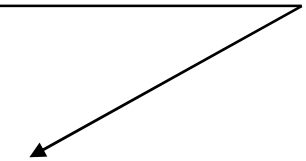
```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments_random(arguments): ...
if __name__ == '__main__':
  ...
```

```
if __name__ == '__main__':
  scenario = sys.argv[1]
  if scenario == 'random':
    ...get parameters...
    ...create grid...
    ...fill grid with random values...
    ...fill to edge...
    ...report...
  else:
    fail('Unknown scenario "%s"' % scenario)
```

```
if __name__ == '__main__':
  scenario = sys.argv[1]
  if scenario == 'random':
    ...do a real simulation...
  elif scenario == '5x5_line':
    ...simulate a run for the edge on a 5x5 grid...
  else:
    fail('Unknown scenario "%s"' % scenario)
```

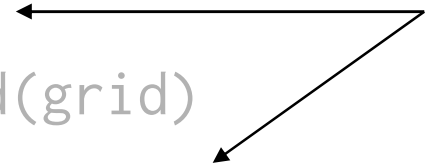
```
...
elif scenario == '5x5_line':
  grid = create_grid(5)
  init_grid_5x5_line(grid)
  num_filled_cells = fill_grid(grid)
  check_grid_5x5_line(grid, num_filled_cells)
...
```

```
...
elif scenario == '5x5_line':
  grid = create_grid(5)
  init_grid_5x5_line(grid)
  num_filled_cells = fill_grid(grid)
  check_grid_5x5_line(grid, num_filled_cells)
...
```



We have these

```
...
elif scenario == '5x5_line':
  grid = create_grid(5)
  init_grid_5x5_line(grid)
  num_filled_cells = fill_grid(grid)
  check_grid_5x5_line(grid, num_filled_cells)
...
```



Must write these

```
...
elif scenario == '5x5_line':
  grid = create_grid(5)
  init_grid_5x5_line(grid)
  num_filled_cells = fill_grid(grid)
  check_grid_5x5_line(grid, num_filled_cells)
...
```

Must write these

Must write a similar pair of functions for each test

```
...
elif scenario == '5x5_line':
  grid = create_grid(5)
  init_grid_5x5_line(grid)
  num_filled_cells = fill_grid(grid)
  check_grid_5x5_line(grid, num_filled_cells)
...
```

Must write these

Must write a similar pair of functions for each test

Write the first pair, then think about refactoring

```
def init_grid_NxN_line(grid):
  '''Fill NxN grid with straight line to edge
  for testing purposes.'''

  N = len(grid)
  for x in range(N):
    for y in range(N):
        grid[x][y] = 2

  for i in range(N/2 + 1):
      grid[N/2][i] = 1
```

```
def init_grid_NxN_line(grid):
    '''Fill NxN grid with straight line to edge
    for testing purposes.'''

    N = len(grid)
    for x in range(N):
        for y in range(N):
            grid[x][y] = 2

    for i in range(N/2 + 1):
        grid[N/2][i] = 1
```

Just as easy to make it general

```
def init_grid_NxN_line(grid):
    '''Fill NxN grid with straight line to edge
    for testing purposes.'''

    N = len(grid)
    for x in range(N):
        for y in range(N):
            grid[x][y] = 2

    for i in range(N/2 + 1):
        grid[N/2][i] = 1
```

This part is easy to understand

```
def init_grid_NxN_line(grid):
    '''Fill NxN grid with straight line to edge
    for testing purposes.'''

    N = len(grid)
    for x in range(N):
        for y in range(N):
            grid[x][y] = 2

    for i in range(N/2 + 1):
        grid[N/2][i] = 1
```

This part is *not* easy to understand

```
def init_grid_NxN_line(grid):
  '''Fill NxN grid with straight line to edge
  for testing purposes.'''

  N = len(grid)
  for x in range(N):
    for y in range(N):
      grid[x][y] = 2

  for i in range(N/2 + 1):
    grid[N/2][i] = 1
```

This part is *not* easy to understand

Bugs in test cases are just more work

```
def init_grid_NxN_line(grid):
  '''Fill NxN grid with straight line to edge
  for testing purposes.'''

  N = len(grid)
  for x in range(N):
    for y in range(N):
      grid[x][y] = 2

  for i in range(N/2 + 1):
    grid[N/2][i] = 1
```

This part is *not* easy to understand

Bugs in test cases are just more work

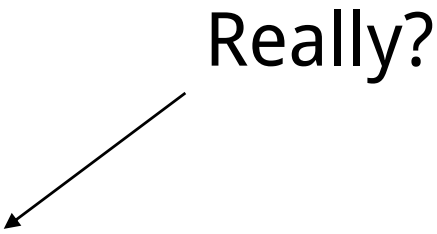
Refactor this later

```
def check_grid_NxN_line(grid, num_filled):
  '''Check NxN grid straight line grid.'''

  N = len(grid)
  assert num_filled == N/2 + 1, 'Wrong number filled'

  for x in range(N):
    for y in range(N):
      if (x == N/2) and (y <= N/2):
        assert grid[x][y] == FILLED, 'Not filled!'
      else:
        assert grid[x][y] != FILLED, 'Wrongly filled!'
```

```
def check_grid_NxN_line(grid, num_filled):
  '''Check NxN grid straight line grid.'''

  N = len(grid)
  assert num_filled == N/2 + 1, 'Wrong number filled'

  for x in range(N):
    for y in range(N):
      if (x == N/2) and (y <= N/2):
        assert grid[x][y] == FILLED, 'Not filled!'
      else:
        assert grid[x][y] != FILLED, 'Wrongly filled!'
```

Slight generalization

```
def check_grid_NxN_line(grid, num_filled):
  '''Check NxN grid straight line grid.'''

  N = len(grid)
  assert num_filled == N/2 + 1, 'Wrong number filled'

  for x in range(N):
    for y in range(N):
      if (x == N/2) and (y <= N/2):
        assert grid[x][y] == FILLED, 'Not filled!'
      else:
        assert grid[x][y] != FILLED, 'Wrongly filled!'
```



```
def check_grid_NxN_line(grid, num_filled):
  '''Check NxN grid straight line grid.'''

  N = len(grid)
  assert num_filled == N/2 + 1, 'Wrong number filled'

  for x in range(N):
    for y in range(N):
      if (x == N/2) and (y <= N/2):
        assert grid[x][y] == FILLED, 'Not filled!'
      else:
        assert grid[x][y] != FILLED, 'Wrongly filled!'
```

Really?

Are we sure?

These two functions are actually correct

These two functions are actually correct

And report that fill_grid behaves properly

These two functions are actually correct

And report that fill_grid behaves properly

But writing and checking two functions like this for each test won't increase our confidence in our program

These two functions are actually correct

And report that fill_grid behaves properly

But writing and checking two functions like this for each test won't increase our confidence in our program

Because the tests are likely to contain bugs

These two functions are actually correct

And report that fill_grid behaves properly

But writing and checking two functions like this for each test won't increase our confidence in our program

Because the tests are likely to contain bugs

We need a simpler way to create and check tests

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
−1	−1	−1	2	2
2	2	2	2	2
2	2	2	2	2

...should fill in like this

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
−1	−1	−1	2	2
2	2	2	2	2
2	2	2	2	2

...should fill in like this

Why not draw our test cases?

```
fixture_5x5_line = '''2 2 2 2 2
                    2 2 2 2 2
                    1 1 1 2 2
                    2 2 2 2 2
                    2 2 2 2 2'''
```

Easy to read

```
fixture_5x5_line = '''2 2 2 2 2
                    2 2 2 2 2
                    1 1 1 2 2
                    2 2 2 2 2
                    2 2 2 2 2'''
```

Easy to read

Easy to write

```
fixture_5x5_line = '''2 2 2 2 2
                    2 2 2 2 2
                    1 1 1 2 2
                    2 2 2 2 2
                    2 2 2 2 2'''
```

Easy to read

Easy to write

So there's no reason not to create lots of tests

```
fixture_5x5_line = '''2 2 2 2 2
                    2 2 2 2 2
                    1 1 1 2 2
                    2 2 2 2 2
                    2 2 2 2 2'''
```

Technical term for

"the thing a test is run on"

Easy to read

Easy to write

So there's no

reason not to

create lots of tests

```
fixture_5x5_line = '''2 2 2 2 2
                    2 2 2 2 2
                    1 1 1 2 2
                    2 2 2 2 2
                    2 2 2 2 2'''

result_5x5_line  = '''. . . . .
                    . . . . .
                    * * * . .
                    . . . . .
                    . . . . .'''
```

Also easy to write and read

```
fixture_5x5_line = '''2 2 2 2 2
                    2 2 2 2 2
                    1 1 1 2 2
                    2 2 2 2 2
                    2 2 2 2 2'''

result_5x5_line  = '''. . . . .
                    . . . . .
                    * * * . .
                    . . . . .
                    . . . . .'''
```

Also easy to write and read

* means 'filled'

```
fixture_5x5_line = '''2 2 2 2 2
                    2 2 2 2 2
                    1 1 1 2 2
                    2 2 2 2 2
                    2 2 2 2 2'''

result_5x5_line  = '''. . . . .
                    . . . . .
                    * * * . .
                    . . . . .
                    . . . . .'''
```

Also easy to write and read

* means 'filled'

. means 'whatever'

```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]

def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```

```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]
```

Put fixtures and results together so that we can loop over them

```
def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```

```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]

def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```

Re-check everything

```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]

def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```

For each test in turn...

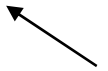
```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]

def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```

Turn the multi-line string into a grid

```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]

def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```

Run the code under test

```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]

def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```



Check it

```
TESTS = [
  [fixture_5x5_line, result_5x5_line],
  ...
]

def run_tests():
  '''Run all the tests at once.'''
  for (fixture, result) in TESTS:
    grid = create_fixture_grid(fixture)
    num_filled = fill_grid(grid)
    check_result_grid(grid, num_filled, fixture, result)
```

Left as an exercise for the viewer

"That's a lot of work"

"That's a lot of work"

Compared to what?

"That's a lot of work"

Compared to what?

- Inspecting printouts of real grids?

"That's a lot of work"

Compared to what?

- Inspecting printouts of real grids?

  - And re-inspecting after every code change?

"That's a lot of work"

Compared to what?

- Inspecting printouts of real grids?

  - And re-inspecting after every code change?

- Or retracting a paper after finding a bug?

"That's a lot of work"

Compared to what?

- Inspecting printouts of real grids?
  - And re-inspecting after every code change?
- Or retracting a paper after finding a bug?

Test code can range from 20% to 200% of application code

"That's a lot of work"

Compared to what?

- Inspecting printouts of real grids?
  - And re-inspecting after every code change?
- Or retracting a paper after finding a bug?

Test code can range from 20% to 200% of application code

(Yes, more test code than application code)

"That's a lot of work"

Compared to what?

- Inspecting printouts of real grids?
  - And re-inspecting after every code change?
- Or retracting a paper after finding a bug?

Test code can range from 20% to 200% of application code

(Yes, more test code than application code)

But that's no different from physical experiments

There are frameworks to help you do this

There are frameworks to help you do this

- We'll look at some in future lectures

There are frameworks to help you do this

- We'll look at some in future lectures

Once tests are written, changing the program itself becomes easier

There are frameworks to help you do this

- We'll look at some in future lectures

Once tests are written, changing the program itself becomes easier

- We'll look at *that* in the next episode

created by

Greg Wilson

June 2010