

Red, Green, Refactor

Creating the Game of Life with Test-Driven Development

By Ken Snyder

Workshop Format

1. Work in pairs
2. Take turns writing tests / code
3. Download Presentation PDF
<http://intel.to/golslides>

3-Minute Setup

- Go to <http://intel.to/gol>
- Download or git checkout the repository
- Follow the instructions in README.md

About the Game of Life

- The Game of Life is a cellular simulation created by British mathematician John Horton Conway.
- After defining an initial state, cells evolve into still lifes, gliders, and long-lived patterns of chaos.
- Cells are placed on a grid according to a given starting state. In each generation, the rules for cell birth and survival are governed by two rules

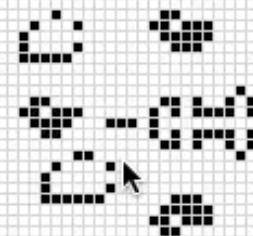
Video 1 – Random Board



Why the Game of Life?

- Architecture is not immediately obvious
- New to many programmers
- It's interesting!
- Play at ConwayJS.com or download Golly

Video 2 – Slow Puffer



2 Reasons To Live

1. Any live cell with 2 or 3 live neighbors survives to the next generation.
2. Any dead cell with exactly 3 live neighbors is born in the next generation.

Other cells die.

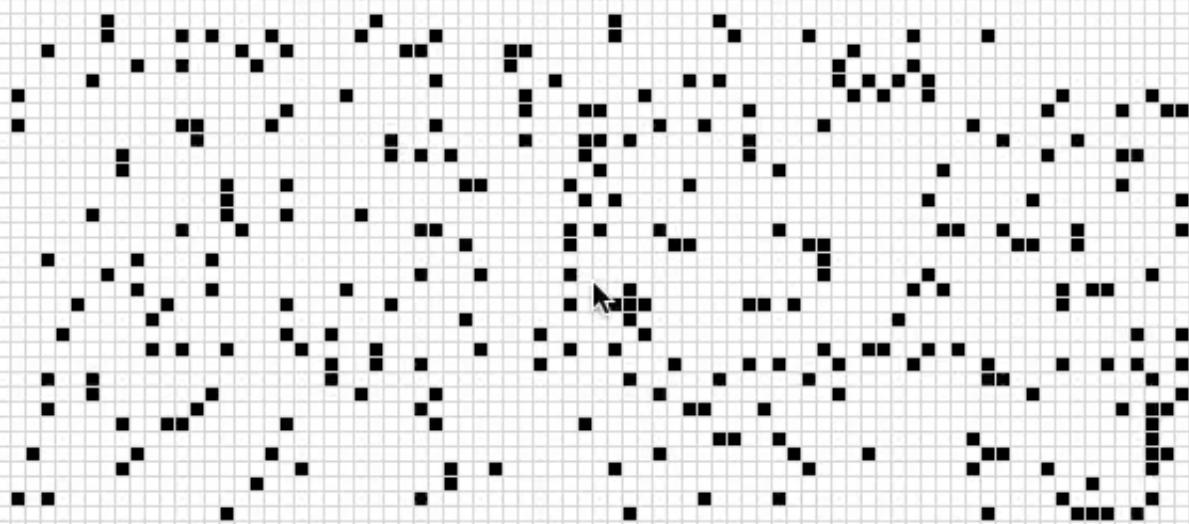
(This ruleset is notated by 23/3)

Metaphor for dying cells



Underpopulation

Video 3 – Maze Rules (12345/3)



QUnit

<http://qunitjs.com>

OR

```
$ npm install grunt-contrib-qunit
```

```
test("Test Name", function() {
    strictEqual(actual, expected, [message]);
    deepEqual(actual, expected, [message]);
    throws(function, [expected], [message]);
    expect(numAssertions);
});
```

QUnit

<http://qunitjs.com>

OR

```
$ npm install grunt-contrib-qunit
```

```
test("Test Name", function() {
    strictEqual(actual, expected, [message]);
    deepEqual(actual, expected, [message]);
    throws(function, [expected], [message]);
    expect(numAssertions);
});
```

NodeUnit

```
$ npm install nodeunit
```

OR

```
$ npm install grunt-contrib-nodeunit
```

```
exports["Test Name"] = function(test) {
  test.strictEqual(actual, expected, [message]);
  test.deepEqual(actual, expected, [message]);
  test.throws(function, [expected], [message]);
  test.expect(numAssertions);
};
```

Other Frameworks

1. Mocha
2. Jasmine
3. jsUnity
4. Testem



The Grid

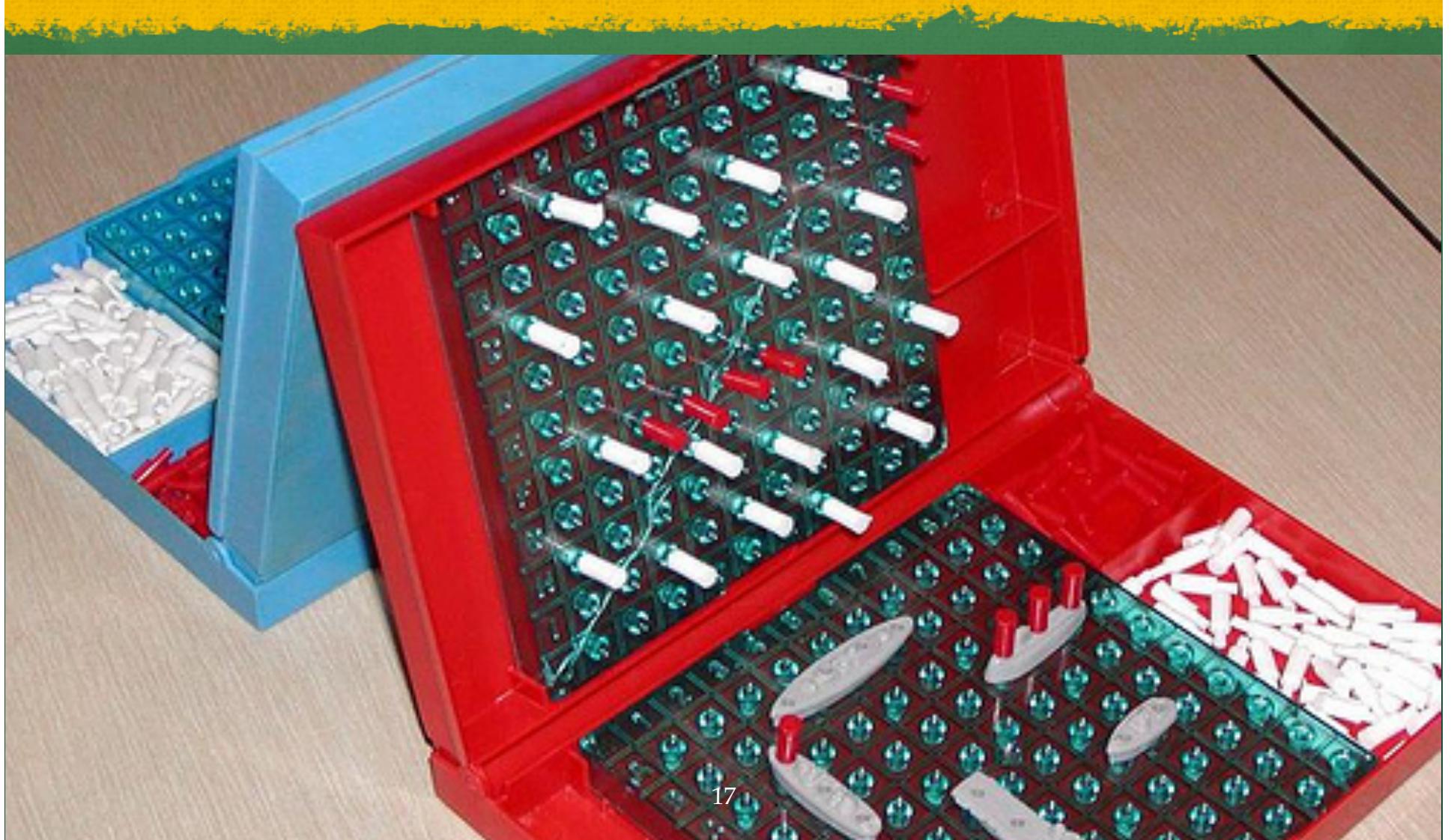
-1,-1	0,-1	1,-1	2,-1	3,-1
-1,0	0,0	1,0	2,0	3,0
-1,1	0,1	1,1	2,1	3,1
-1,2	0,2	1,2	2,2	3,2
-1,3	0,3	1,3	2,3	3,3

**Red includes
Thinking**

**I DON'T ALWAYS USE
MEMES**



1f: Call functions before they exist



1p: Write as little code as possible



1p: Write documentation first



2p: Delay implementation



3p: Start with a naïve implementation



3p: Avoid architecture too early



4p: Write code for one case at a time



6f: Reset test conditions often



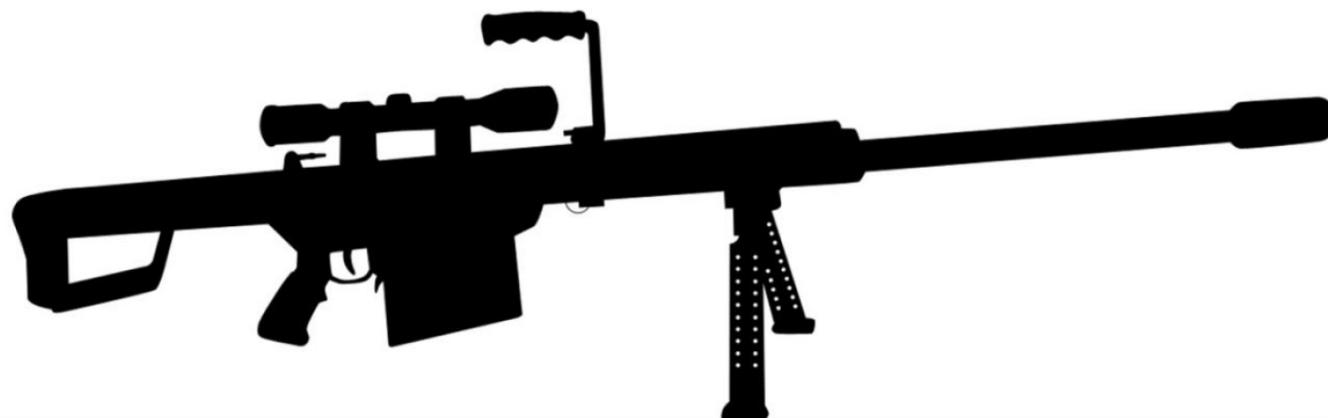
**8af: Don't be
afraid to
switch gears**



8af: Write API to be publicly testable

PRIVATE PROPERTY

**IF YOU CAN READ THIS YOU
ARE WITHIN RANGE**



11f: Using properties is not wrong



12p: Refactor duplication



Other reasons to refactor

- Divide code into smaller pieces
- Divide responsibility intelligently
- Improve readability
- Suggested by code review

BUT:

Don't get refactoring confused with
architecture

Summary

- (1f) Call functions before they exist
- (1p) Write as little code as possible!
- (1p) Write documentation first
- (2p) Delay implementation
- (3p) Start with a naïve implementation
- (3p) Avoid architecture too early
- (4p) Write code for one case at a time

Summary (cont'd)

- (6f) Reset test conditions often
- (8af) Don't be afraid to switch gears
- (8af) Write API to be publicly testable
- (11f) Using properties is not wrong
- (12p) Refactor duplication, etc.

Thanks for Participating

A word cloud centered around the word "code". The words are arranged in a roughly circular pattern around the central word "code". The size of each word indicates its frequency or importance in the context of the slide. The words include: test, code, function, unit, testing, tests, example, DOM, NodeUnit, QUnit, assertion, expected, existing, output, update, ago, ad-hoc, date, initial, see, change, output, Refactoring, first, just, useful, assertions, JavaScript, changes, method, enough, new, look, something, argument, failed, results, page, elements, passed, element, called, using, actual, framework, result, element, equal, like, problem, first, just, useful, assertions, and date.

test

code

function

unit

testing

tests

example

DOM

NodeUnit

QUnit

assertion

expected

existing

output

update

ago

ad-hoc

date

initial

see

change

output

Refactoring

first

just

useful

assertions

JavaScript

changes

method

enough

new

look

something

argument

failed

results

page

elements

passed

element

called

using

actual

framework

result

element

equal

like

problem

first

just

useful

assertions

date