

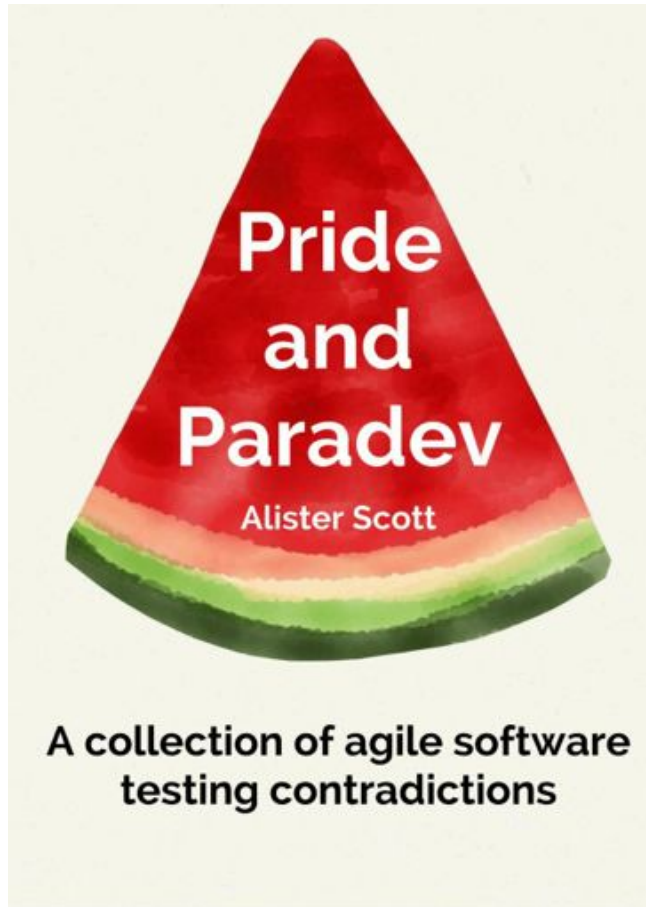
Einstein

building a minesweeper robot using tests

Alister Scott

ThoughtWorks®

About Me



- QA consultant @ ThoughtWorks Brisbane
- Tester/Coder
- Father of 2 (almost 3) boys
- Blogger of **watirmelon.com**
- Writer of '**Pride & Paradev**'
- Collector of arid plants

**what can writing a
minesweeper robot
teach us about testing?**

**who here has never
played minesweeper?**



**who here has never
heard of
minesweeper?**

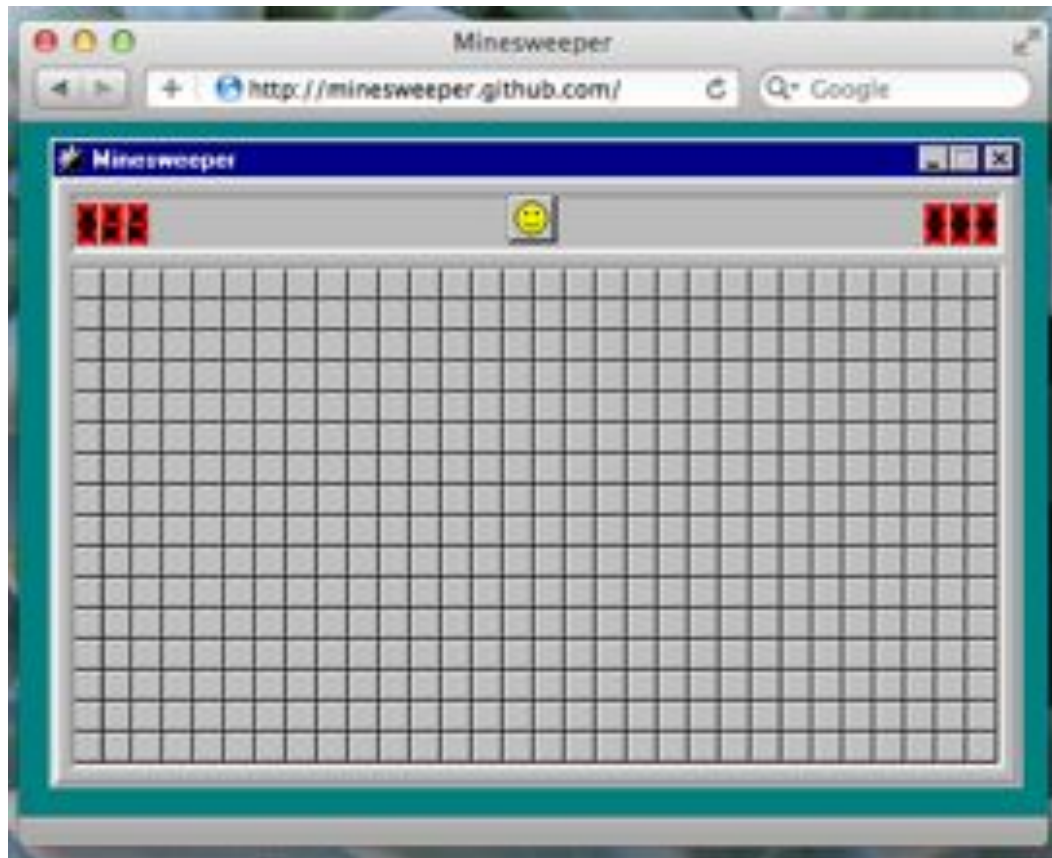
a little refresher
<video>

windows
minesweeper has
some issues:

- * non testable**
- * hard to automate**
- * need windows**

we wrote our own

minesweeper.github.io



open source
CoffeeScript

347 lines of
CoffeeScript =>
478 lines of
JavaScript

minesweeper.github.io

Jasmine specs

74 specs in 0.3
seconds

687 lines of
CoffeeScript
(nearly double
prod code)



The screenshot shows the Jasmine test runner interface. At the top, it displays 'Jasmine 1.1.0 revision 1315677058' and 'Show passed skipped'. Below this, a summary bar indicates '74 specs, 0 failures in 0.312s' and 'Finished at Mon Mar 12 2012 12:05:35 GMT+1000 (EST)'. The main content area lists 14 individual specs for the 'minesweeper' suite, all of which are marked as 'run'.

Spec Name	Status
minesweeper	run
should display test mode when specifying mine locations	run
should not display test mode when using random mines	run
should cycle through marked to uncertain to unclicked on right click	run
should reveal cell with no adjacent mines	run
should reveal cell with one adjacent mine	run
should reveal cell with two adjacent mines	run
should reveal cell with three adjacent mines	run
should reveal cell with four adjacent mines	run
should reveal cell with five adjacent mines	run
should reveal cell with six adjacent mines	run
should reveal cell with six adjacent mines	run
should reveal cell with six adjacent mines	run
should reveal adjacent cells when there are no adjacent mines	run
should display depressed button when indicator button is clicked	run

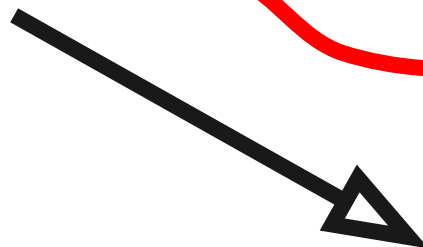
**biggest benefit of
writing something
yourself is building
testability in**

**but writing the game
minesweeper is so
much easier than
writing a robot to play
it => I'll talk about that**

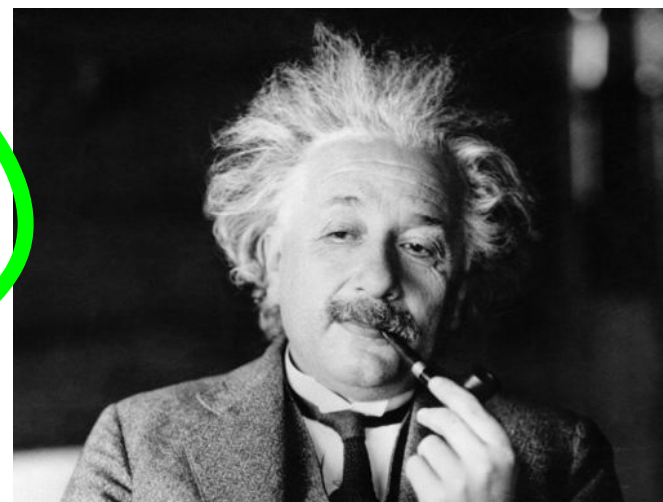
**solving a complex
problem (automating
minesweeper) is a lot
like testing; I'll share
my lessons on the way**



how?



tests!



how do we create an Einstein robot?

outside



in

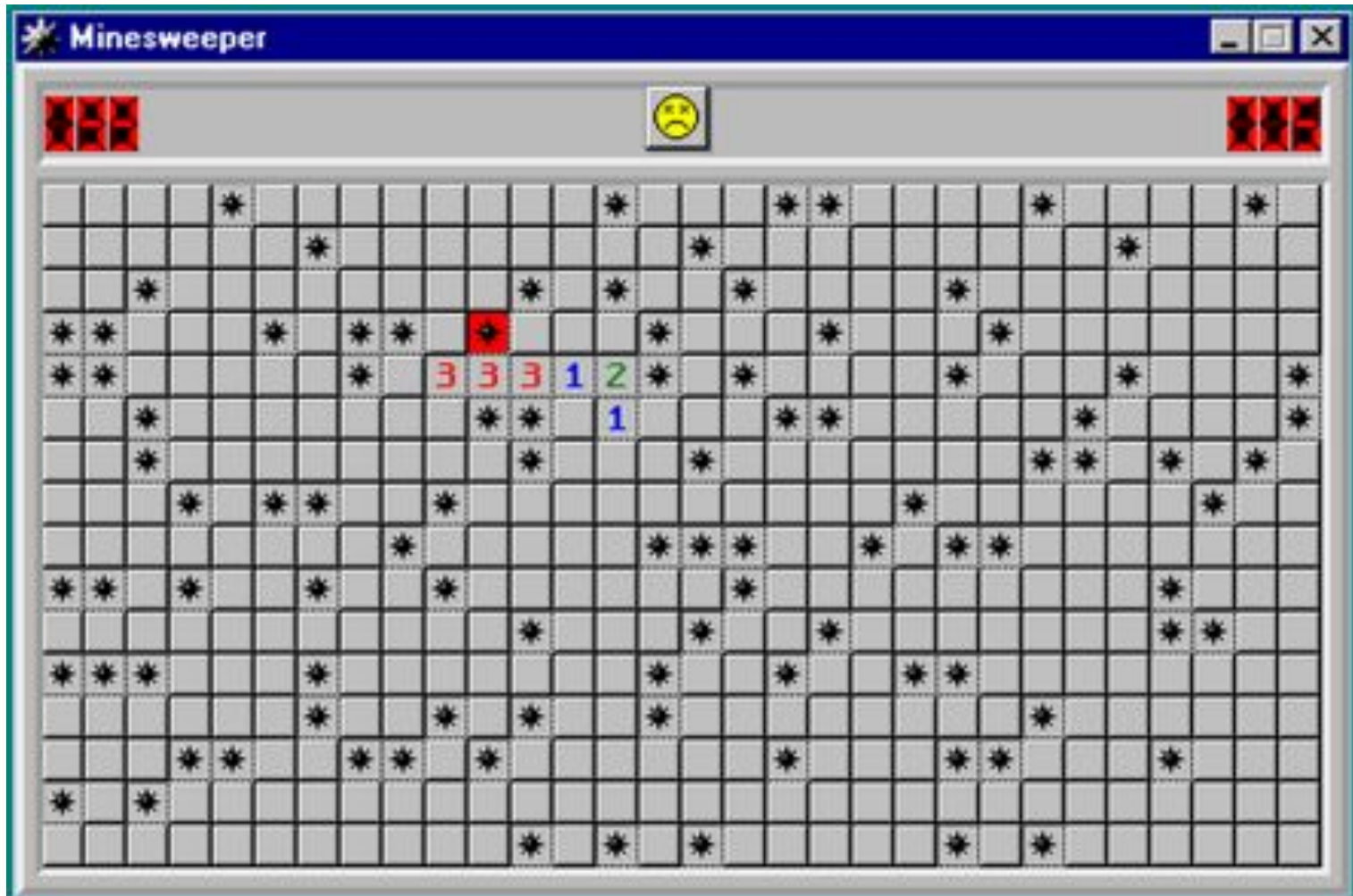


Break it down

"The secret of getting ahead is getting started. The secret of getting started is breaking your complex overwhelming tasks into small manageable tasks, and then starting on the first one."

~ Mark Twain

iteration one: guess

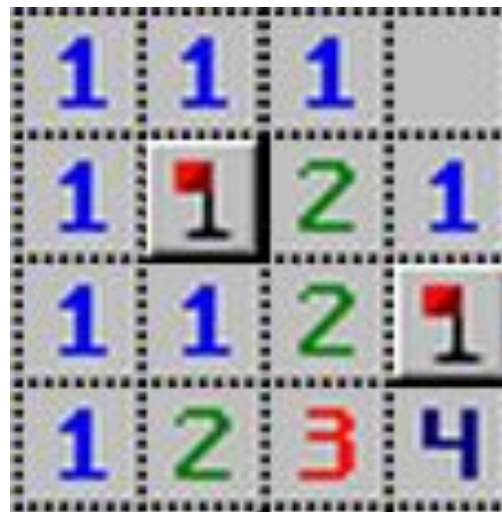


success rate: < 0.1%



**how can you break
your big testing
problems down into
smaller ones and get
started?**

iteration two: identify obvious mines

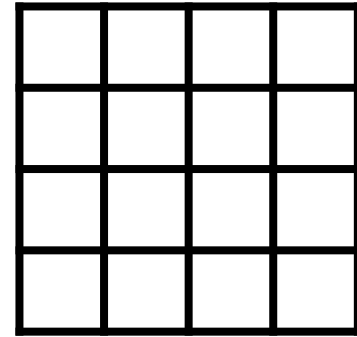
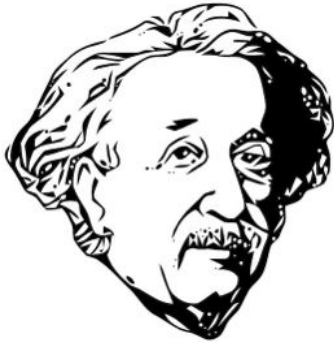


iteration two: identify obvious mines

```
describe 'safe cells to click' do
  it 'should determine safe cells to left click' do
    analyse <<-EOF
    * 1
    . .
    EOF
    analyser.safe_cells_to_click.should == [[1,0],[1,1]]
  end
end
```

success rate: < 2%

a minesweeper robot: separation of concerns



can play any minesweeper
implementation game using in
built intelligence

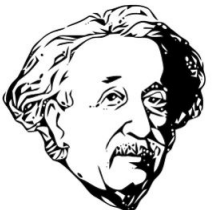
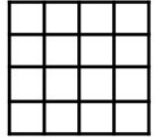
a web implementation of
minesweeper that can be
automatically controlled

automated unit tests using
RSpec w/ mocks

automated functional tests
using Cucumber & Watir-
Webdriver

automated functional tests using Cucumber & Watir-Webdriver

1 feature: 4 scenarios
35 seconds



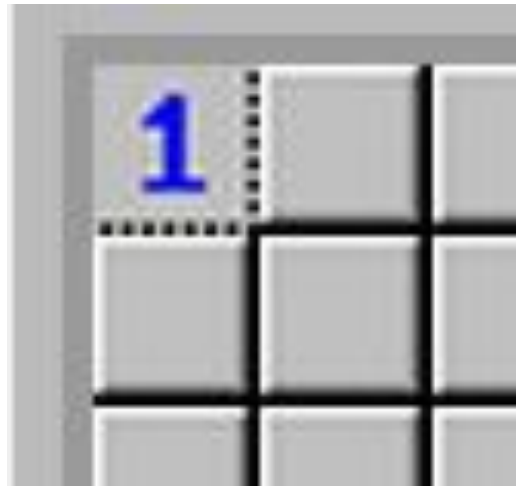
automated unit tests using RSpec w/ mocks

48 specs in 0.2 seconds

minesweeper robot testing pyramid

**are there other ways
that you can test?
perhaps accessing an
API or via the
database?**

iteration three: guess better



```
it 'should calculate mine placement probability considering adjacent revealed cells'  
  analyse <<-EOF, 1  
  1 . .  
  . . .  
  EOF  
  analyser.probability_of_mine_at(0,1).should be_within(0.001).of(0.333)  
end
```

success rate: 5%



iteration four: 'clusters'

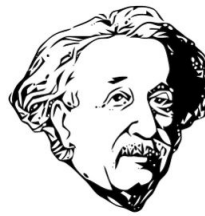
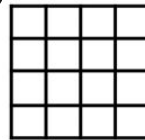
```
it 'should detect a safe cell to click taking an adjacent  
cell cluster into consideration' do  
  analyse <<-EOF  
  0 0 0 .  
  1 1 2 .  
  . . . .  
  EOF  
  analyser.safe_cells_to_click.should == [[2,2]]  
end
```

success rate: 13%

iteration five: 'derived clusters'
via exploratory testing

success rate: 20%

automated functional
tests using Cucumber &
Watir-Webdriver



automated unit tests
using RSpec w/ mocks

exploratory testing

**every guess that
shouldn't be a guess
is a bug (which can
kill us)**

Exploratory Testing

1. Let robot play heaps of games
2. Study each guess
(logs/screenshots)
3. Was it absolutely necessary to guess? If not, it's a **bug!**
4. Write failing spec from screenshot
5. Make it **pass!**

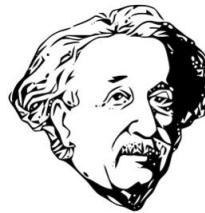
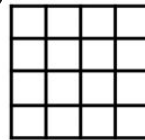
eliminate unnecessary guesses!



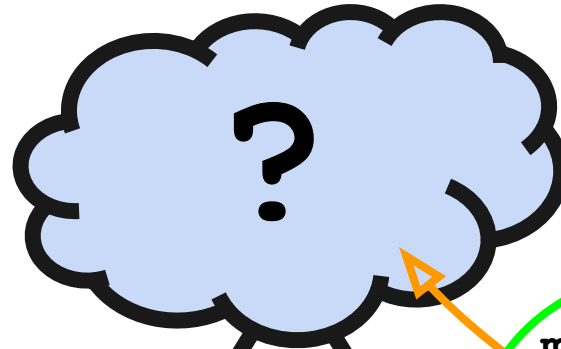
```
describe 'safe cells to click' do
  it 'should detect safe cells taking multiple non-
  adjacent cell clusters into consideration' do
    analyse <<-EOF
    3 * 3 1 2 1
    * * 4 * 3 *
    3 4 . . . 2
    * 2 . 2 . 1
    EOF
    analyser.safe_cells_to_click.should == [[3, 2],
      [3, 4], [2, 3]]
  end
end
```

example: gets hard

automated functional
tests using Cucumber &
Watir-Webdriver



automated unit tests
using RSpec w/ mocks



**manual exploratory
testing:**
scan logs

look at screenshots on
guesses

exploratory testing

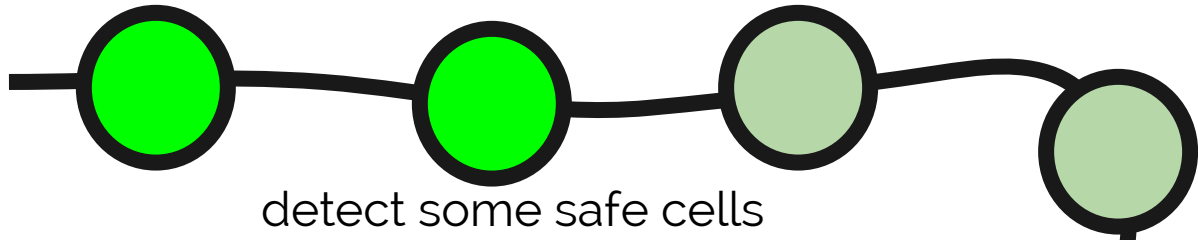
**how can you use
exploratory testing to
ensure your
automated testing is
valid?**

how can you use

logs?

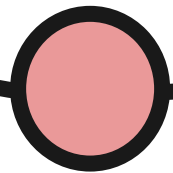
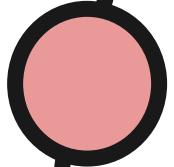
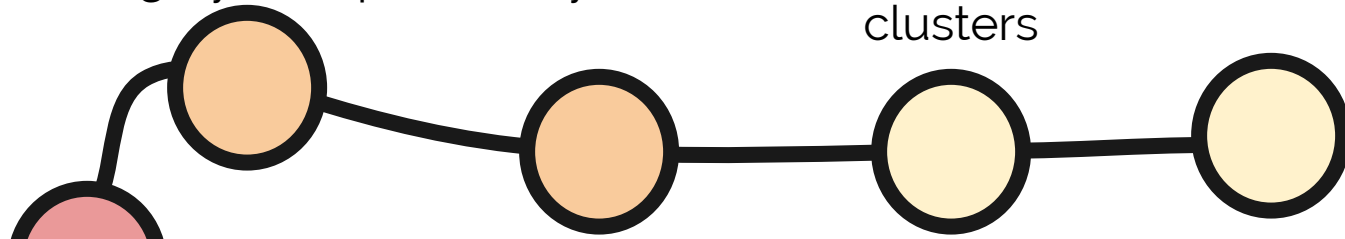


detect some obvious mines



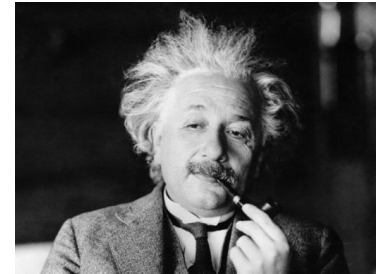
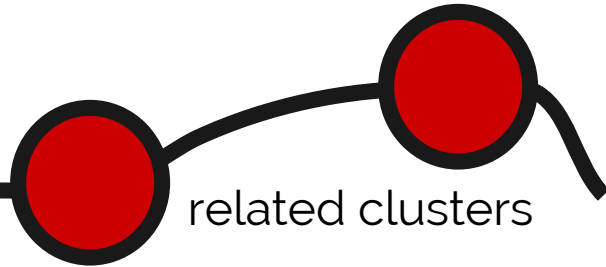
guessing by field probability

clusters



derived clusters

related clusters



progressive intelligence: law of diminishing returns?

**have you noticed as
you do more and
more testing, it gets
harder to find bugs?**

Performance Testing

How do you test performance of an algorithm that tries to solve a **non-deterministic** problem?



Can always win this without guessing (start at 0,0)

Scenario: The user is efficient at playing expert games

Given the field

```

. . . . . * * . . . * . . . . . * . . . . .
. . . * . * . . . . . * . . . * . . . * * * * . . .
. . . . . . . . . . . * * * . . . * * . . . * . . .
* . . . * * * . . . * . . . . . . . . . . . * . . .
. . . . . . . . . . . * . . . * . . . . . * . . . *
* . . . . . . . . . * * . . . . . * . . . * . . . *
. . . . . . . . . . . * * . . . * . . . . . * . . . *
* * * . . . . . * . . . . . . . . . * . . . * . . . *
* * . . . * * . . . . . * * . . . * . . . * . . . *
. . . * * * * . . . . . . . . . . . . . . . . . *
. . . . . * . . . * . . . * . . . . . * . . . * . . .
. . . * . . . . . * * . . . * . . . * * . . . * . . .
. . . * . . . * * * . . . * * * * . . . * . . . * . . .
. * * * * . . . . . * * * * . . . * . . . . . * . . . *
. . . . . . . . . . . * . . . . . * * . . . * . . . *
*****

```

Then I can win the game without guessing

And I should finish the game in 40 seconds or less

Ensure we never degrade performance

**can you find a known
constant pattern that
you can baseline &
measure
performance against
over time?**

Einstein's Performance Metrics

Beginner

Games won: 85%

Best time: 2 seconds

Intermediate:

Games won: 65%

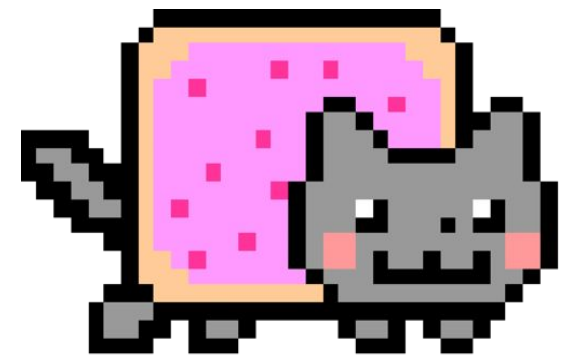
Best time: 8 seconds

Expert:

Games won: 20%

Best time: 21 seconds





Enough talk already

Let's see Einstein in action!

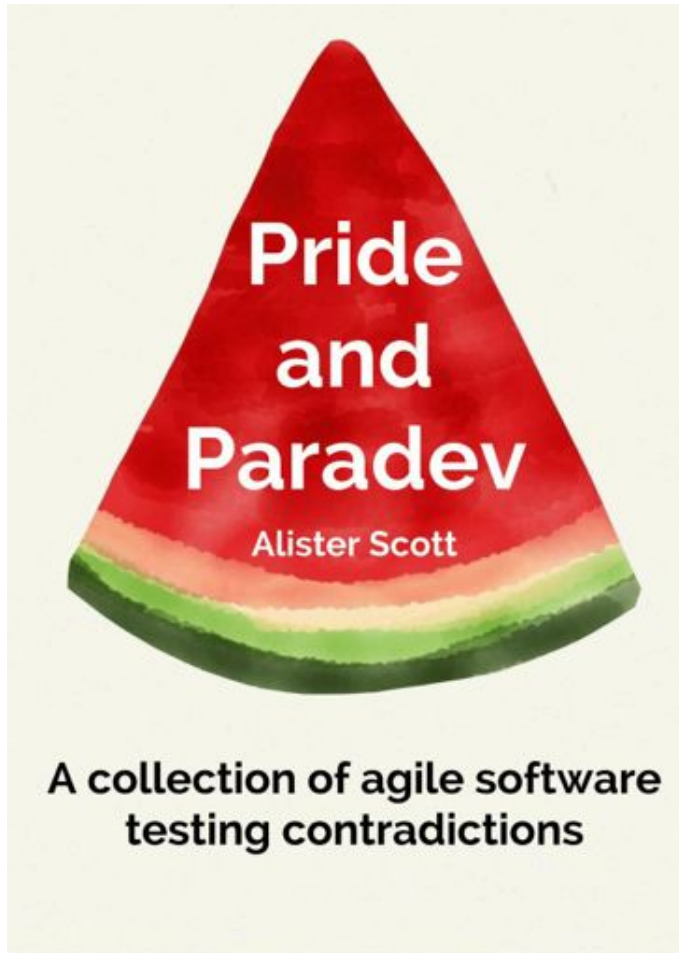
[\[http://soundcloud.com/blaketothefuture/nyan-cat-the-movie\]](http://soundcloud.com/blaketothefuture/nyan-cat-the-movie)

Lessons Learned

- Break big things down into little things
- Design for testability - test only features
- Test in the right places
- Supplement your automated testing with exploratory testing
- Build in performance measures
- Change your mind
- Be young, be foolish, be happy

what's one thing you'll try today?

More



Source code: github.com/minesweeper/minesweeper-robot

Game: minesweeper.github.io

My blog: watirmelon.com

My book: leanpub.com/pride-and-paradev