

# Coderetreat

# Global Day of Coderetreat



Welcome!



**ONE DOES NOT SIMPLY...**

**MISS A CODERRETREAT!**

# Facilitators

→ **Eric Stewart**



twitter: @ericstewart

→ **David Croley**



twitter: @davidcroley

# Schedule

1. About Coderetreat
2. Design
3. Conway's Game of Life
4. Start Sessions

# What Is Coderetreat?



A close-up photograph of a koala's face and upper body. The koala is light gray with dark gray ears and a dark gray nose. It is holding a bright green eucalyptus leaf in its mouth and appears to be chewing on it. Its front paws are gripping a dark brown, textured tree trunk. Some green leaves are visible behind the koala's head.

**SO YOU MEAN  
CODERETREAT**

**WILL IMPROVE MY  
KOALAFICATIONS?**

# **History of Coderetreat**

The idea was spawned at CodeMash Conference '09 and first held on January 25, 2009

Originated by:

- Gary Bernhardt
- Patrick Welsh
- Nyan Hajratwala
- Corey Haines

## **Global Day of Coderetreat**

This year, we have 141 participating cities, with 2,500 expected software developers!

This year's Global Day of Coderetreat is supporting CoderDojo: a worldwide, volunteer-driven movement of programming clubs for young people. Any funds not used to support GDCR will go to CoderDojo.

# **GDCR Supports CoderDojo**

**The global network of free computer programming clubs for young people.**

CoderDojo is a global movement of free, volunteer-led, community based programming clubs for young people.

<http://www.coderdojo.com>





hello@hwf.io

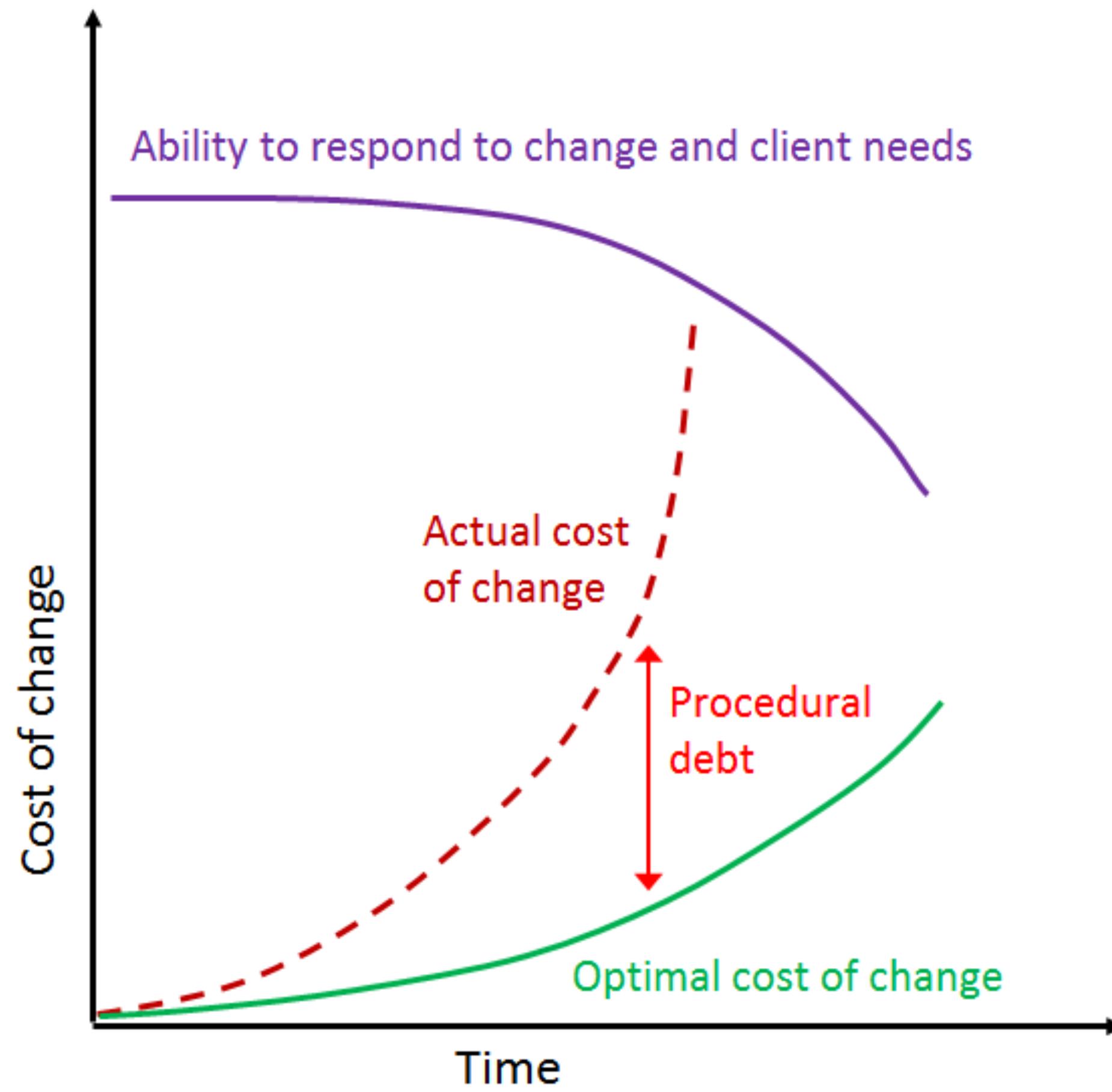
[www.hwf.io](http://www.hwf.io)

## **Sponsors**

- Thank the sponsors at all levels
- Make sure you get discounts sheet!

# Goals

- Perfect Practice and Mastery
  - Learn through Pairing
- Extend your Comfort Zone
  - Experiment
  - Learn new practices
- Think deeply about design
  - Have fun!



## Format

- 8-8:45am: arrive, coffee/breakfast
- 8:45-9am: Welcome, Introductions, Explain the Problem
- 9-Noon: Sessions 1-3
- Noon-1: Lunch, Socializing
- 1-4pm: Sessions 4-6
- 4-4:30pm: Closing circle

# Sessions

- 45 minutes of coding
  - Fresh project
- Work in pairs (decide on language)
  - Apply constraints
  - Don't try to finish!!!
- 15 minutes retrospective/break
  - Delete your code

# Pair Programming



# Pair Programming

- Two people per computer
- Decide on language
- Work together
- **Who** does **what?**





# Pair Programming - If Hollywood Made a Movie

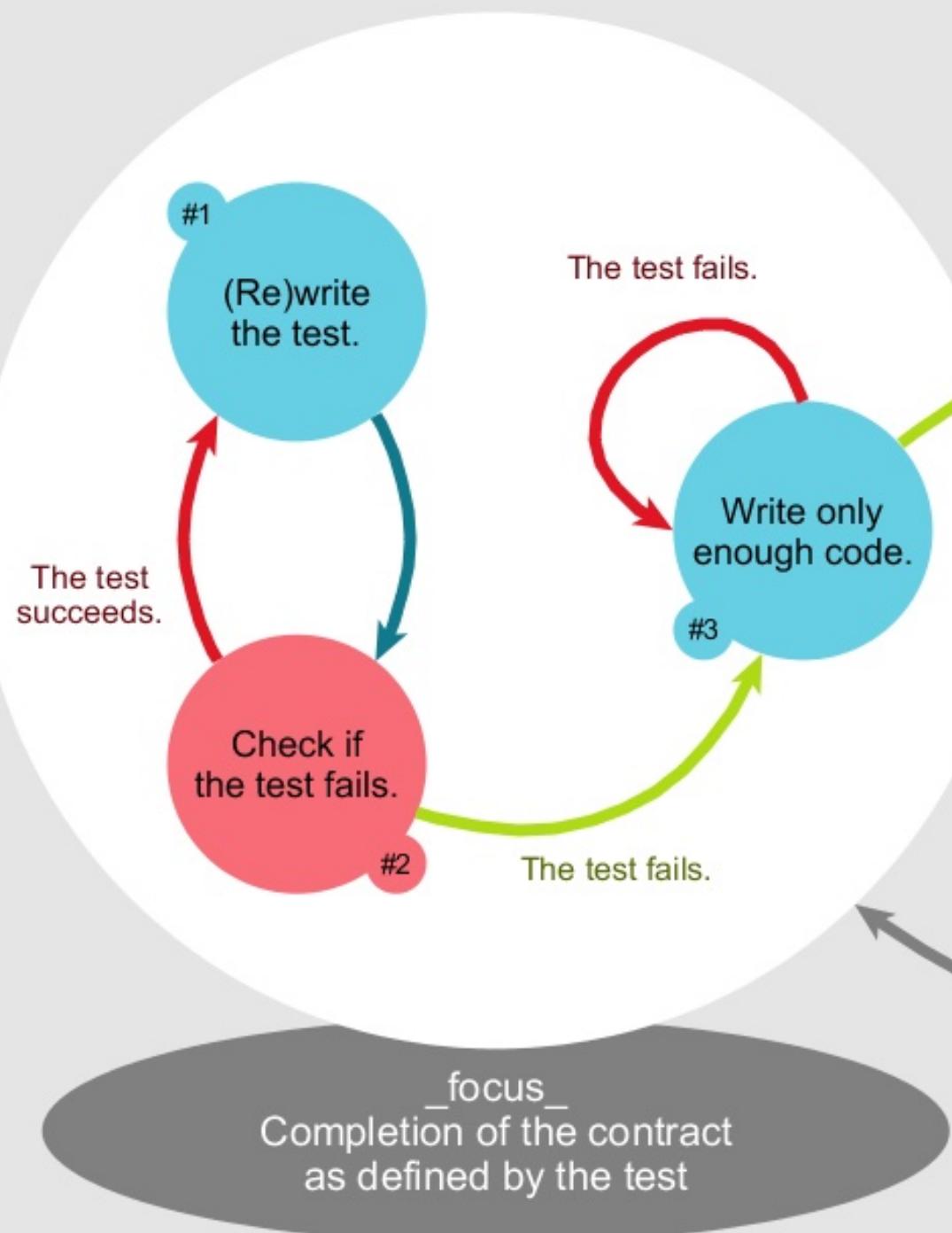


# **Test-Driven Development (TDD)**

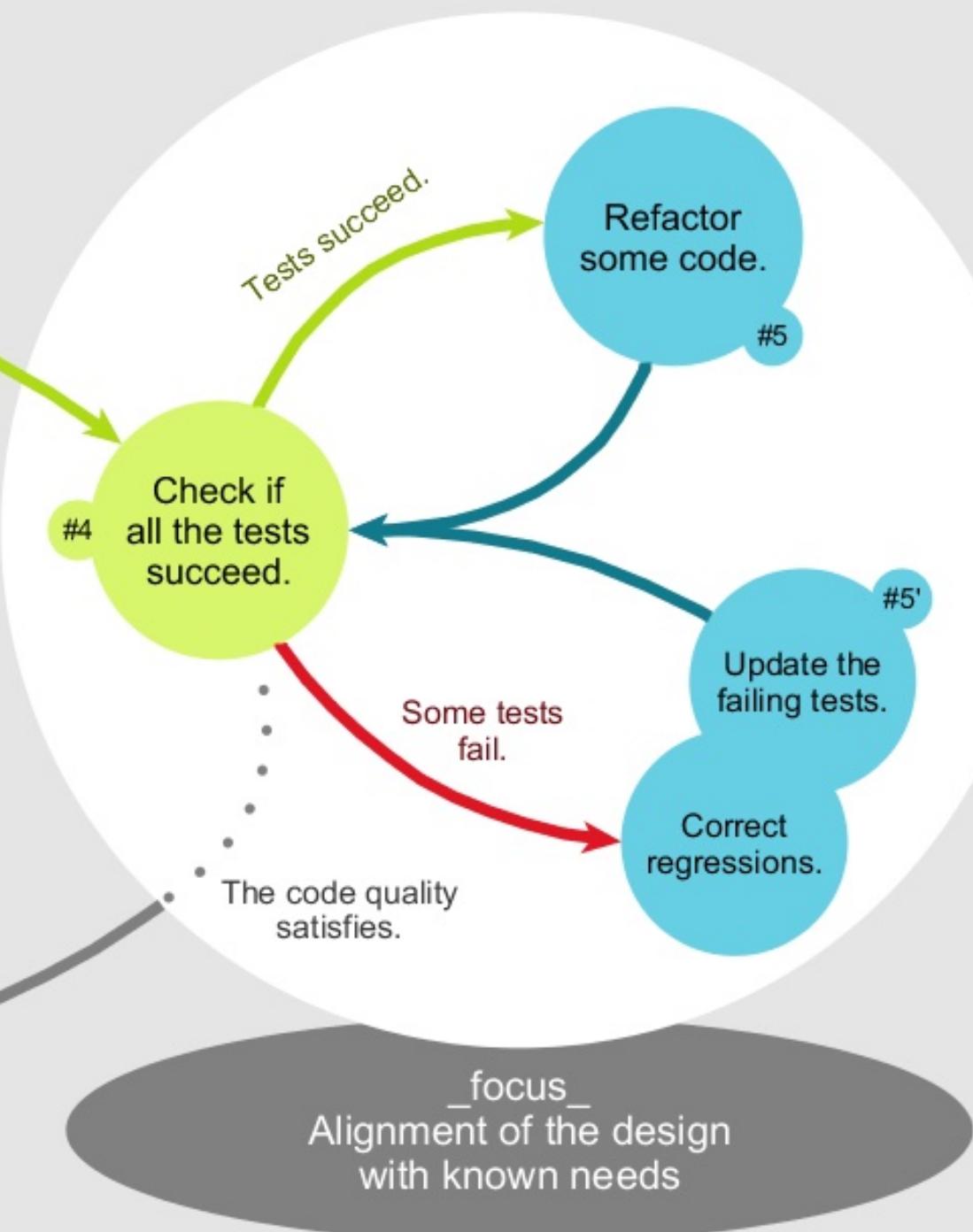
## TDD Lifecycle

1. Add a test
2. Run all tests and see if the new one fails
3. Write some code
4. Run tests
5. Refactor code
6. Repeat

## TEST-FIRST DEVELOPMENT



## REFACTORING



# **Participant Introductions**

- Get a name tag
- Put your name and preferred languages on the whiteboard

# Simple Design

*Perfection is achieved, not  
when there is nothing more  
to add, but when there is  
nothing left to take away.*

— *Antoine de Saint-Exupéry*

*Any intelligent fool can make things bigger, more complex and more violent. It takes a touch of genius and a lot of courage to move in the opposite direction.*

— *Albert Einstein.*

*Don't write code that  
guesses the future.  
Arrange code so that you  
can adapt to the future  
when it arrives.*

— *Sandi Metz*

# **Simple Design**

**Simple Design is one that is easy to change.**

We don't know exactly what is going to need to change. We'll never be more ignorant than we are at this moment.

## The Four Rules Simple Design (Kent Beck)

1. Passes All Its Tests
2. Clearly Expresses Intent
3. Contains No Duplication
4. Small (Has No Superfluous Parts)

## **1. Passes All Tests**

Good design/factoring/etc. is useless if code doesn't do what is intended

## **2. Clearly Expresses Intent**

Pay attention to naming, readability, and how code expresses itself

### **3. No Duplication**

Essentially the DRY (Don't Repeat Yourself) principle of removing knowledge duplication.

## **Don't Repeat Yourself (DRY)**

This rule isn't about code duplication; it is about knowledge duplication:

***Every piece of knowledge should have one and only one representation.***

— *The Pragmatic Programmer*<sup>pragprog</sup>

## **4. Small**

No unused code, check for missing or duplicate abstractions, look for over-abstraction.

Priority



**Passes the Tests**

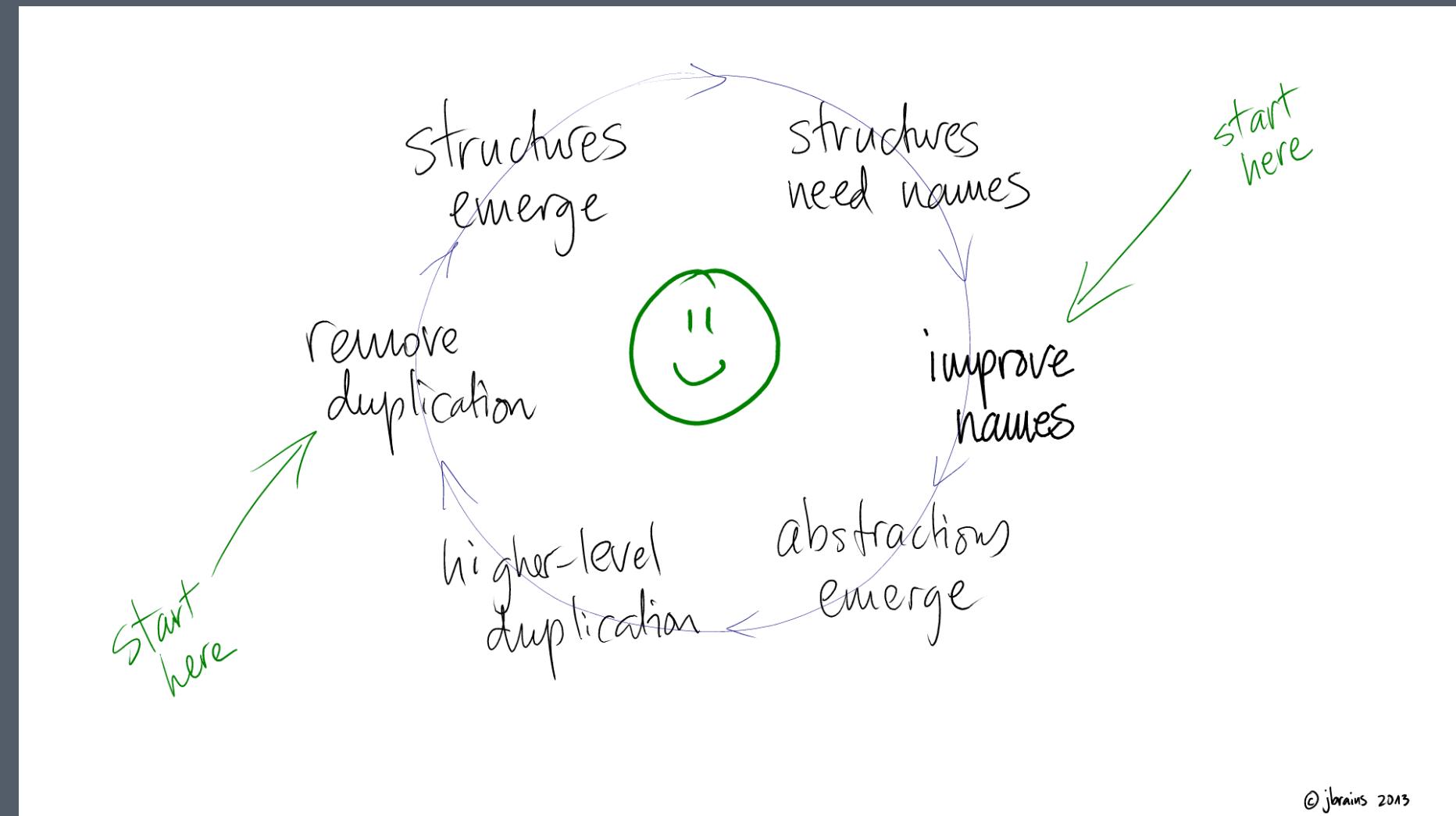
**Reveals Intention**

**No Duplication**

**Fewest Elements**

## J.B. Rainsberger's Simple Design Dynamo

<http://blog.thecodewhisperer.com/2013/12/07/putting-an-age-old-battle-to-rest/>



# SOLID Principles

→ **S**ingle Responsibility Principle

A module should have one, and only one, reason to change.

→ **O**pen-Closed Principle

You should be able to extend a classes behavior without modifying it

→ **L**iskov Substitution Principle

Derived objects must be substitutable for the base classes

→ **I**nterface Segregation Principle

Make fine-grained interfaces that are client specific

→ Dependency inversion principle

Depend on abstractions, not on concretions. Dependency injection is an example.

# **Law of Demeter**

**(Principle of Least Knowledge)**

***A method can access either locally-instantiated variables, parameters passed in, or instance variables.***

Or more simple....

***One dot per statement.***

# Law of Demeter Example<sup>2</sup>

```
class Wallet
  attr_accessor :cash
end
class Customer
  has_one :wallet
end
class Paperboy
  def collect_money(customer, due_amount)
    if customer.wallet.cash < due_amount
      raise InsufficientFundsError
    else
      customer.wallet.cash -= due_amount
      @collected_amount += due_amount
    end
  end
end
end
```

<sup>2</sup> <http://www.dan-manges.com/blog/37>

# Law of Demeter (Improved)<sup>2</sup>

```
class Wallet
  attr_accessor :cash
  def withdraw(amount)
    raise InsufficientFundsError if amount > cash
    cash -= amount
    amount
  end
end
class Customer
  has_one :wallet
  # behavior delegation
  def pay(amount)
    @wallet.withdraw(amount)
  end
end
class Paperboy
  def collect_money(customer, due_amount)
    @collected_amount += customer.pay(due_amount)
  end
end
```

<sup>2</sup> <http://www.dan-manges.com/blog/37>

# Tell, Don't Ask<sup>tda</sup>

It is okay to use accessors to get the state of an object, as long as you don't use the result to make decisions outside the object.

Any decisions based entirely upon the state of one object should be made 'inside' the object itself.

<sup>tda</sup> <http://c2.com/cgi/wiki?TellDontAsk>

# Tell, Don't Ask Example<sup>tda</sup>

```
price = item.Price  
if (price < 20) {  
    item.SetAsBargain()  
}
```

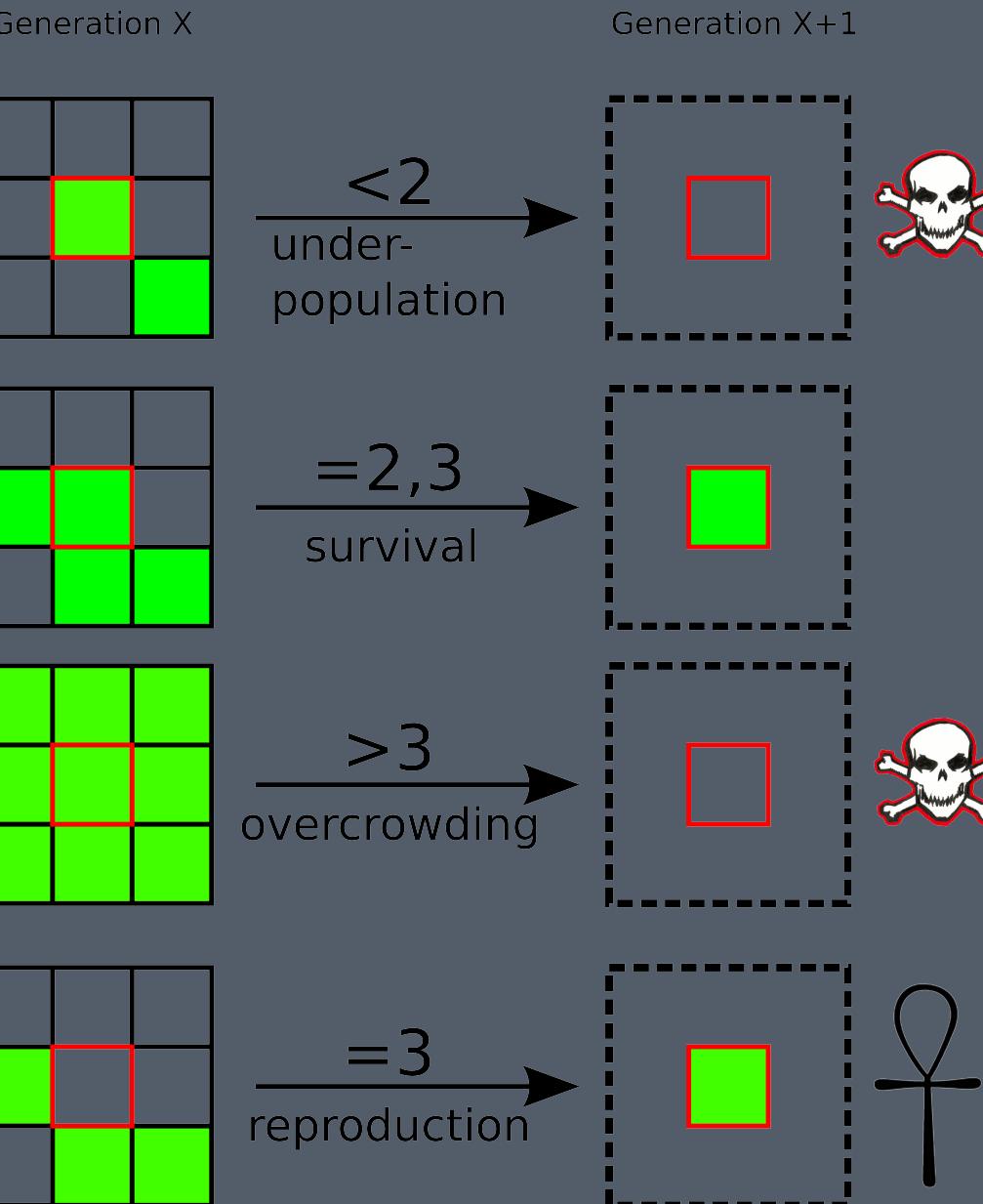
```
price = item.Price  
this.LineColor? = price < 20 ? "Green" : "Red"
```

# Conway's Game of Life

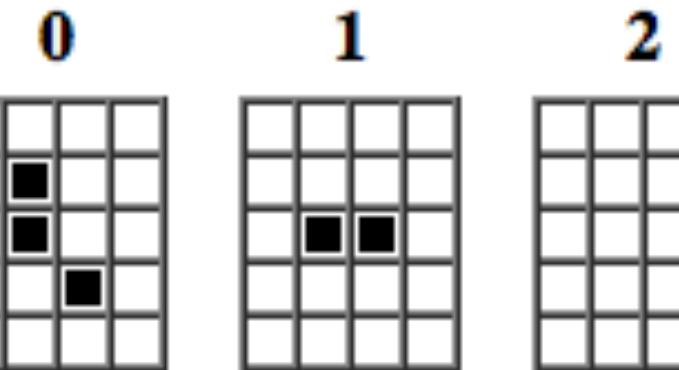




# Game of Life Rules



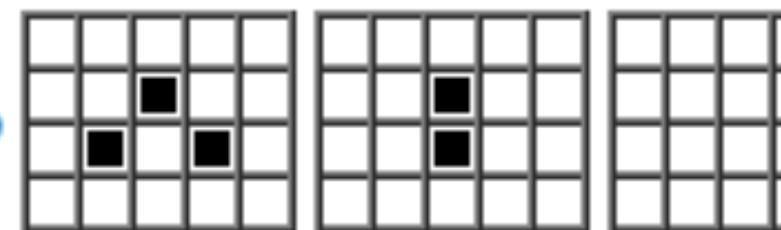
**moves**



**1**



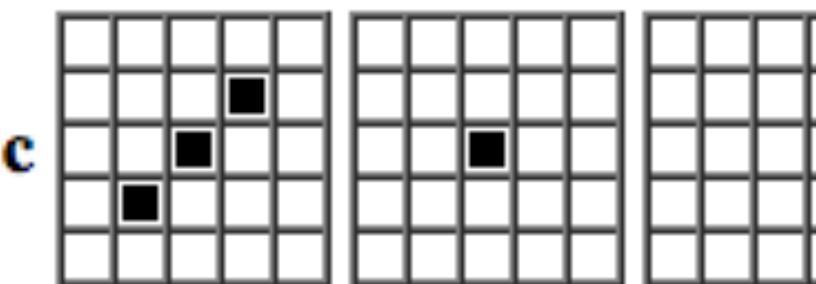
dies



**1**



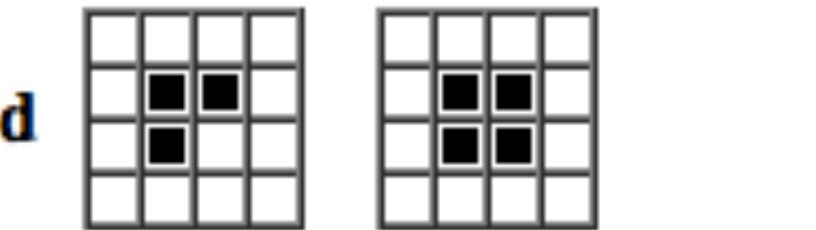
dies



**1**



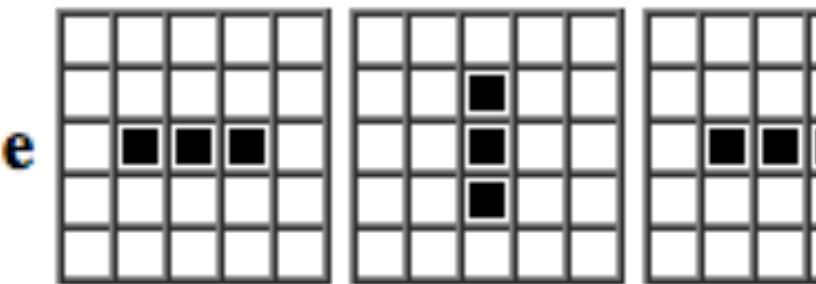
dies



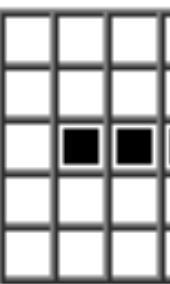
**1**



block (stable)

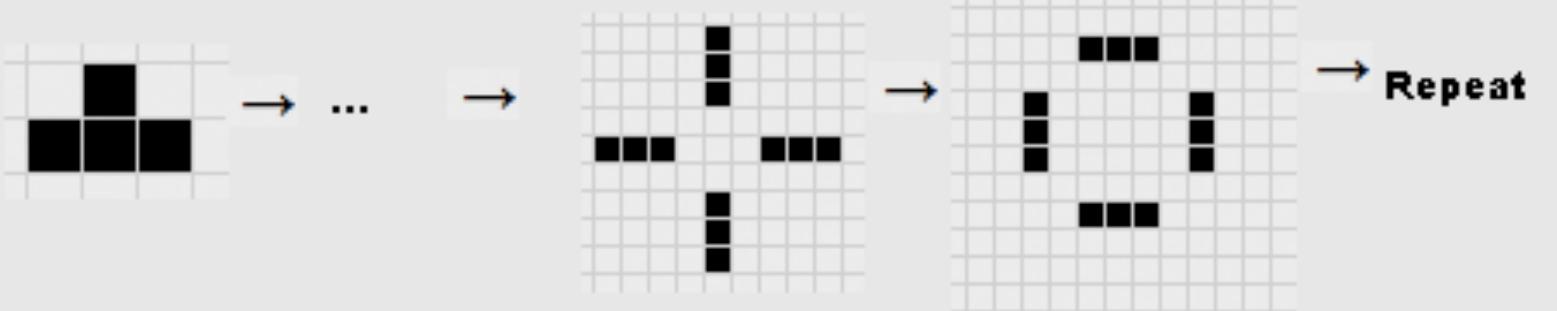
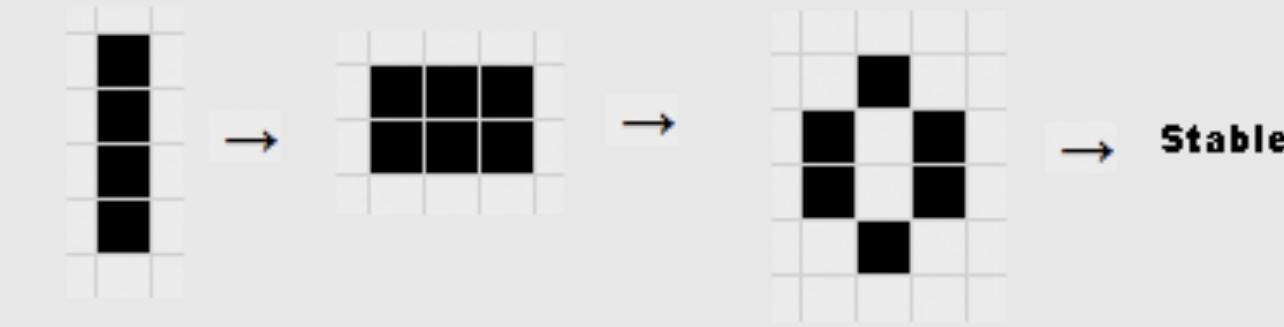
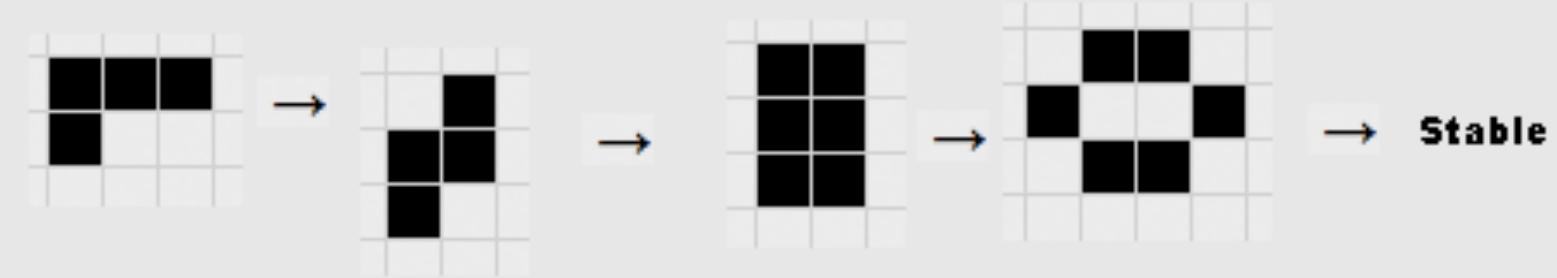


**1**



blinker (period 2)

**Patterns**

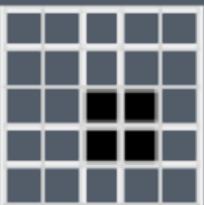


## Tetromino Patterns

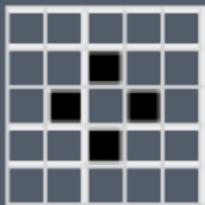
# Still Lifes

Some patterns do not change from one generation to the next

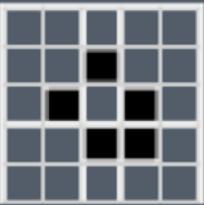
*block*



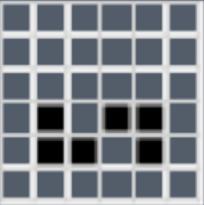
*tub*



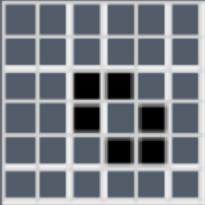
*boat*



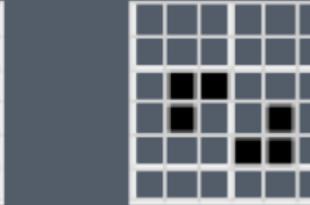
*snake*



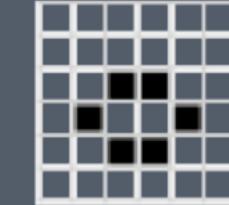
*ship*



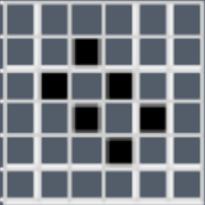
*aircraft carrier*



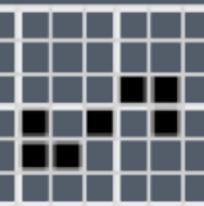
*beehive*



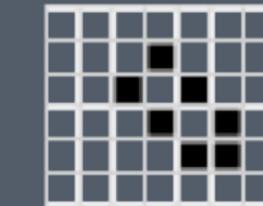
*barge*



*Python*



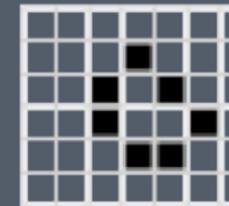
*long boat*



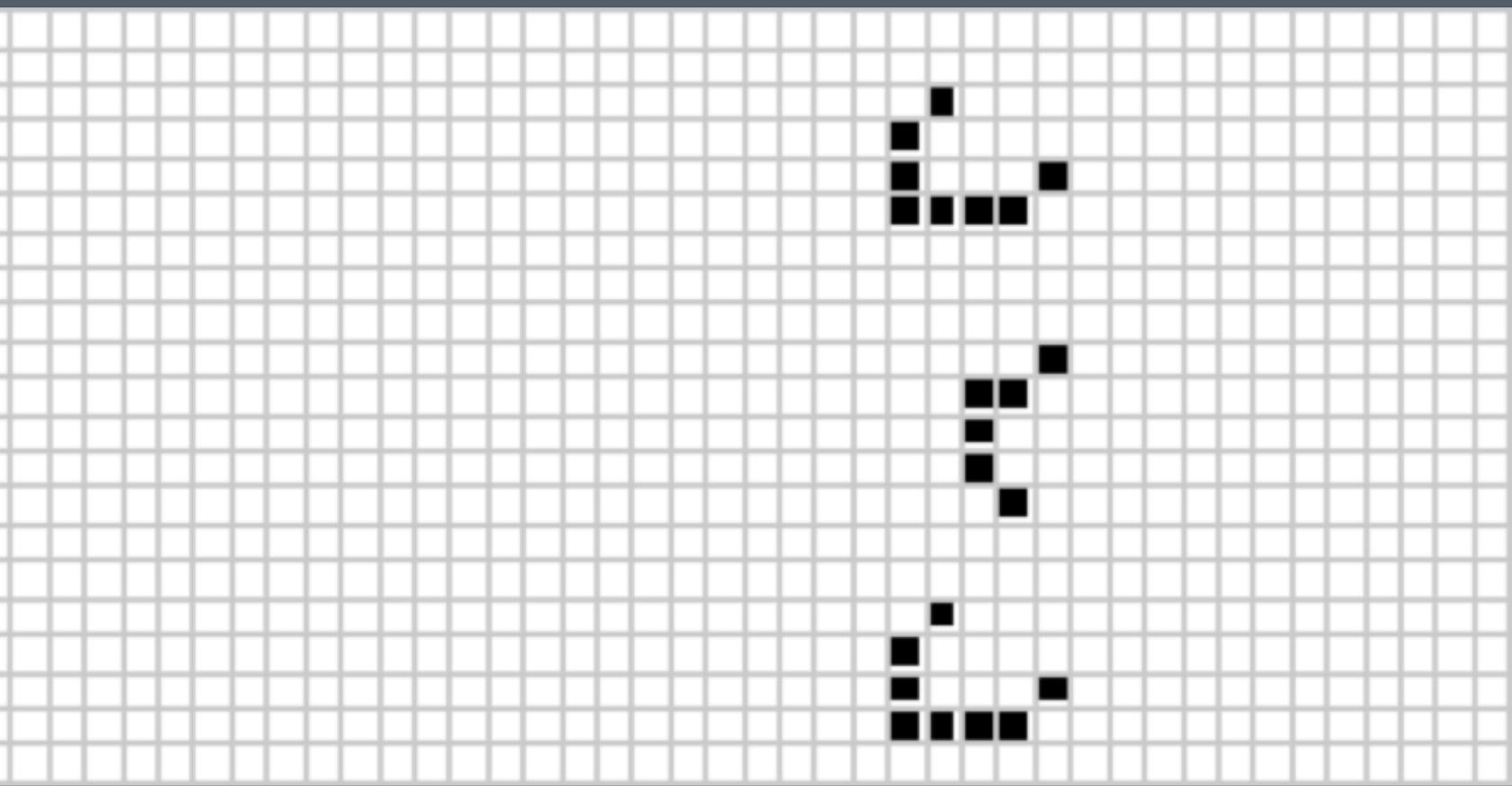
*cater, fishhook*



*loaf*



# Oscillators



# Resources

# Starting Points

- [Github: coreyhaines/coderetreat](#)
- [Github: Agilefreaks/coderetreat](#) - C#, JavaScript, Ruby, Swift

- [Github: justinwyer/coderetreat](#) - Java



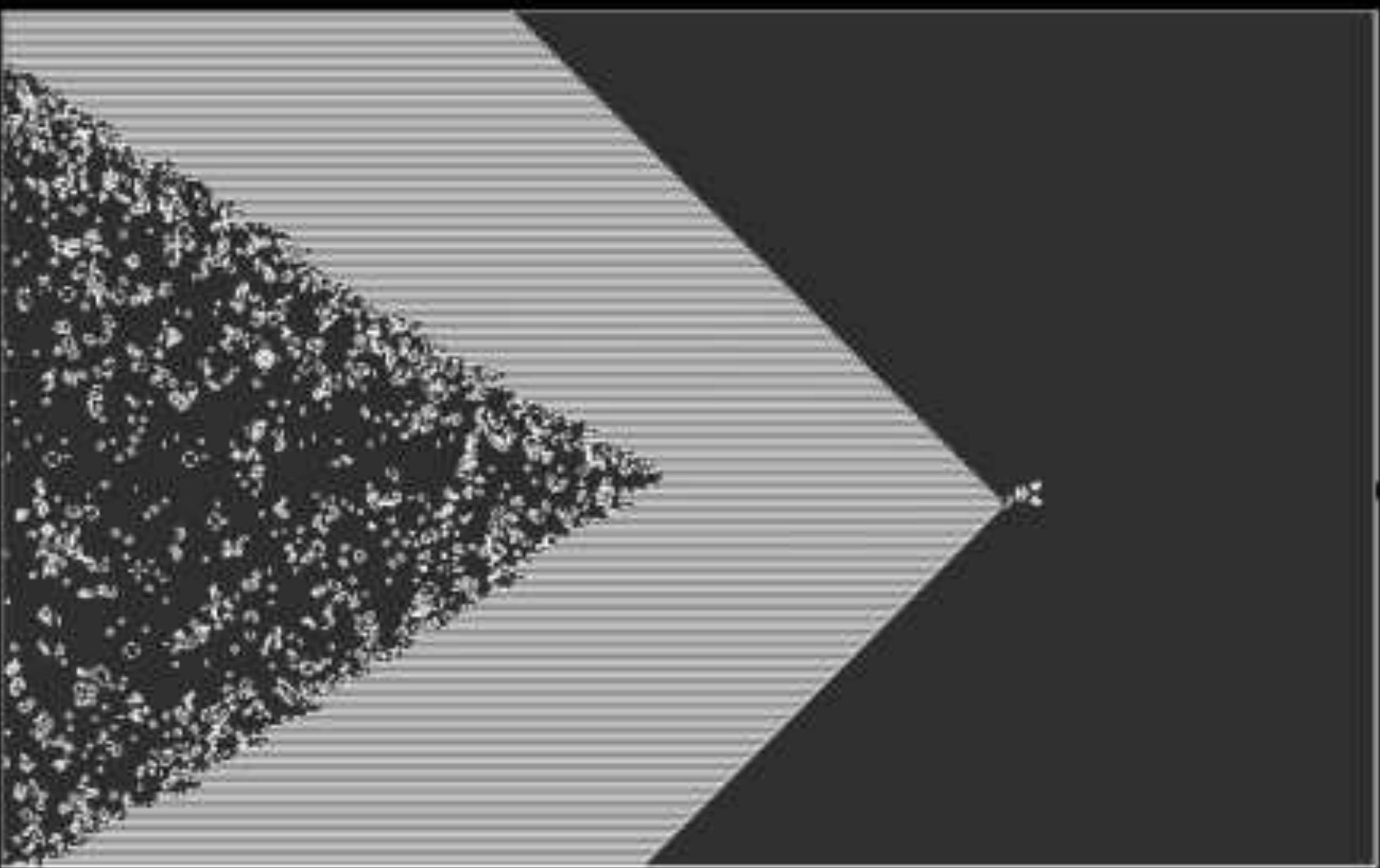
**Web**

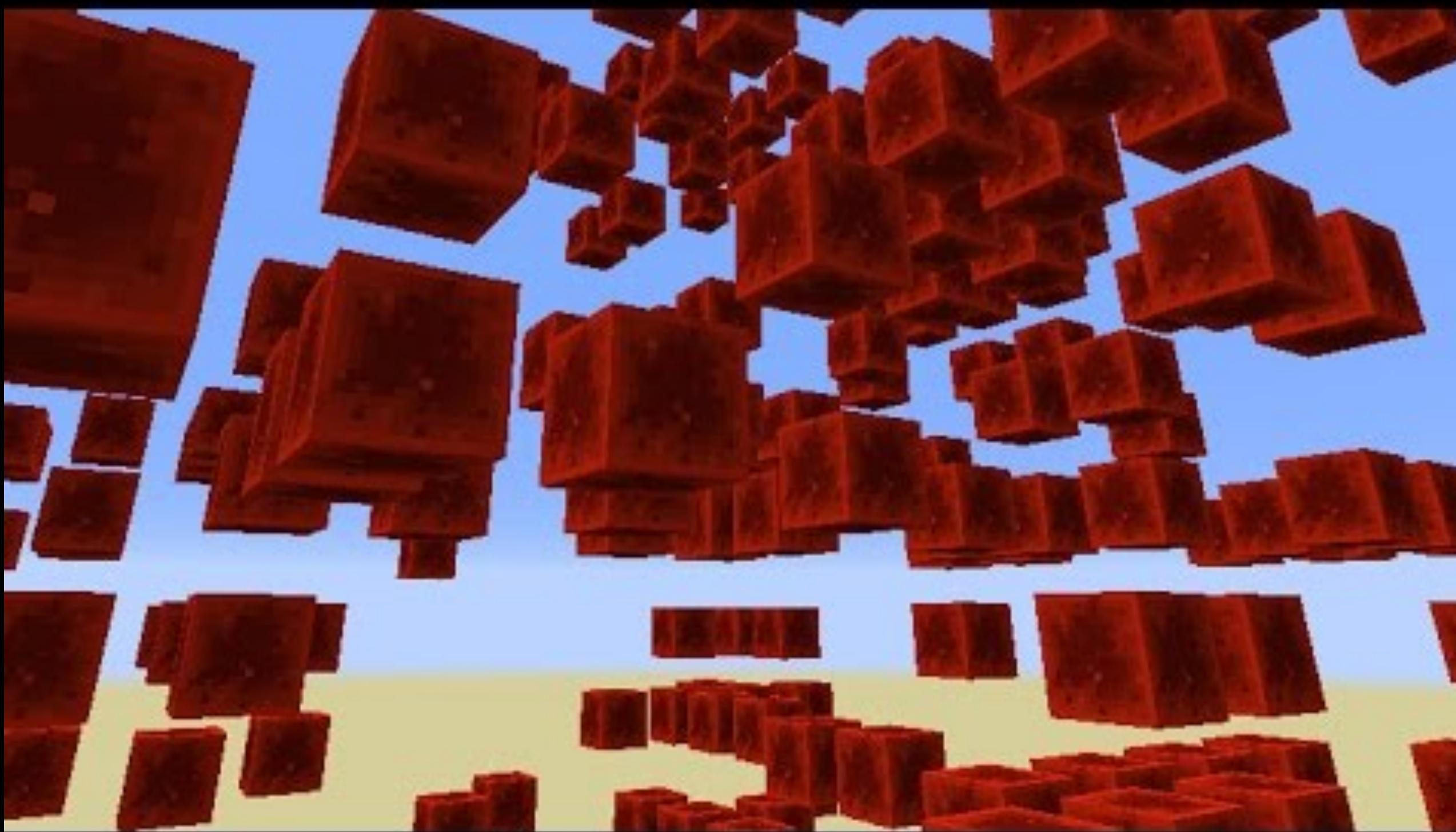
→ LifeWiki

# **Software**

→ Golly

# Videos







# *Conway's Game of Life*



