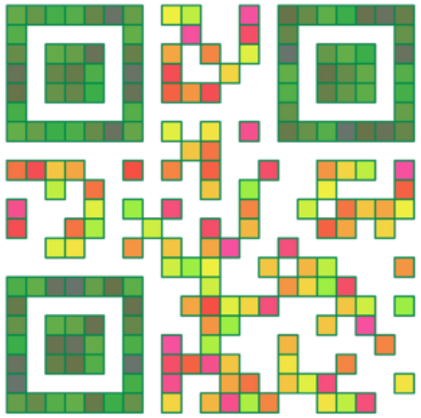
Property Based Testing for Better Code



@jessitron

Outpace™

lots of tests

maintenance burden

didn't test seams

Scala

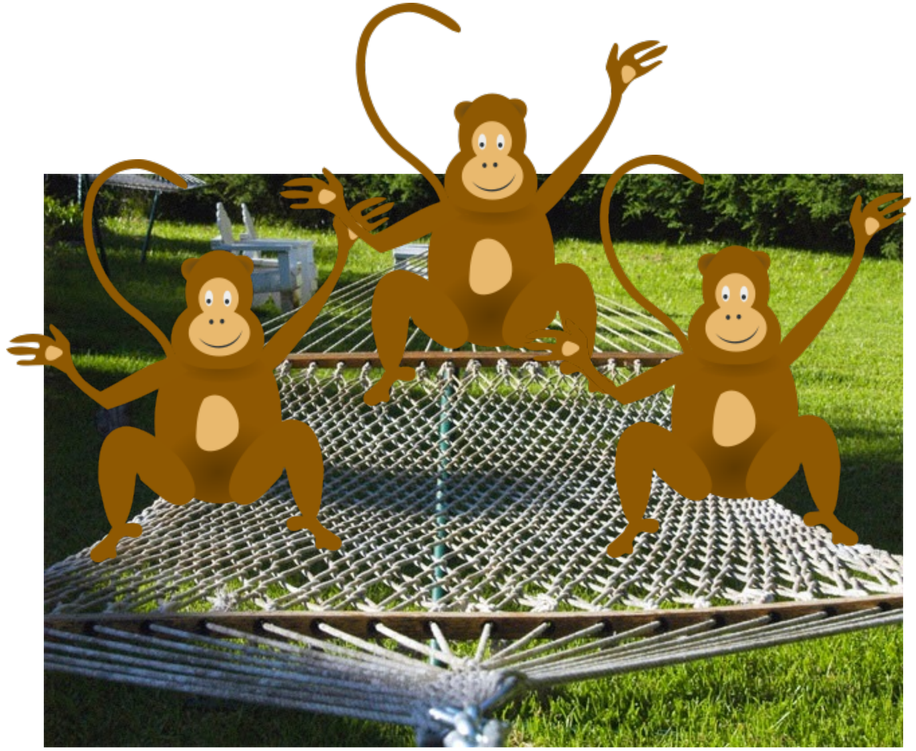
ScalaCheck

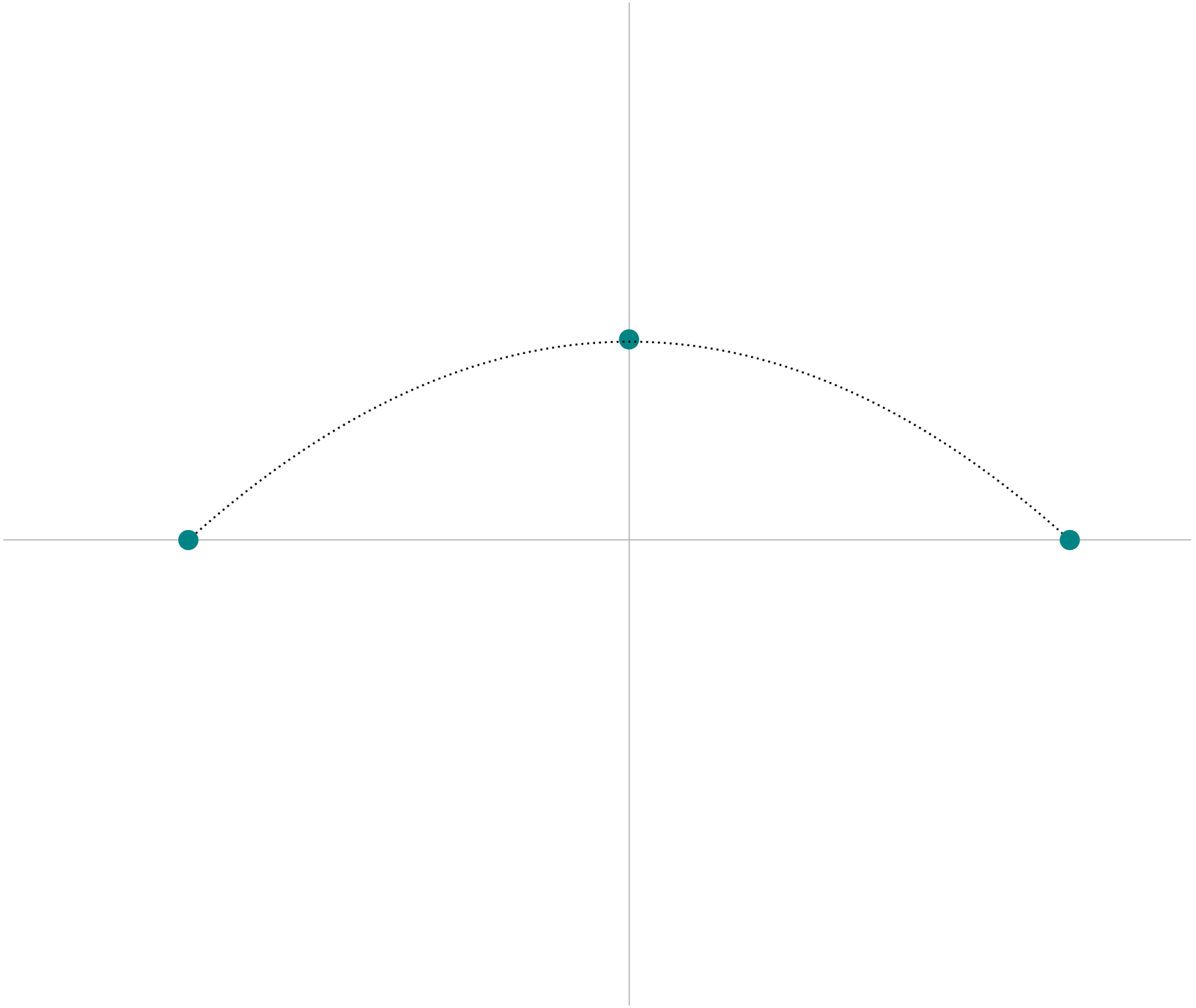
Property Based Testing

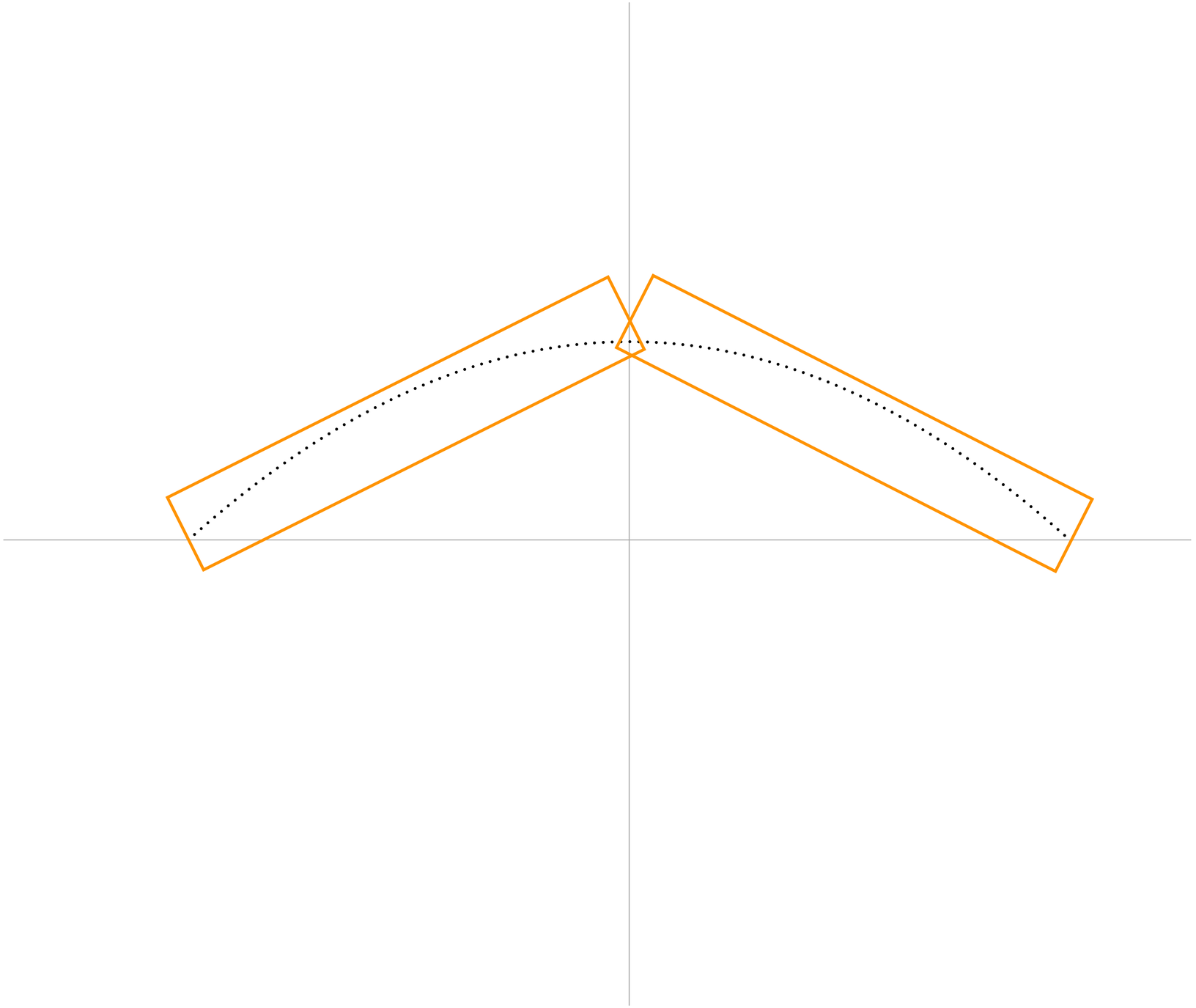
Scala

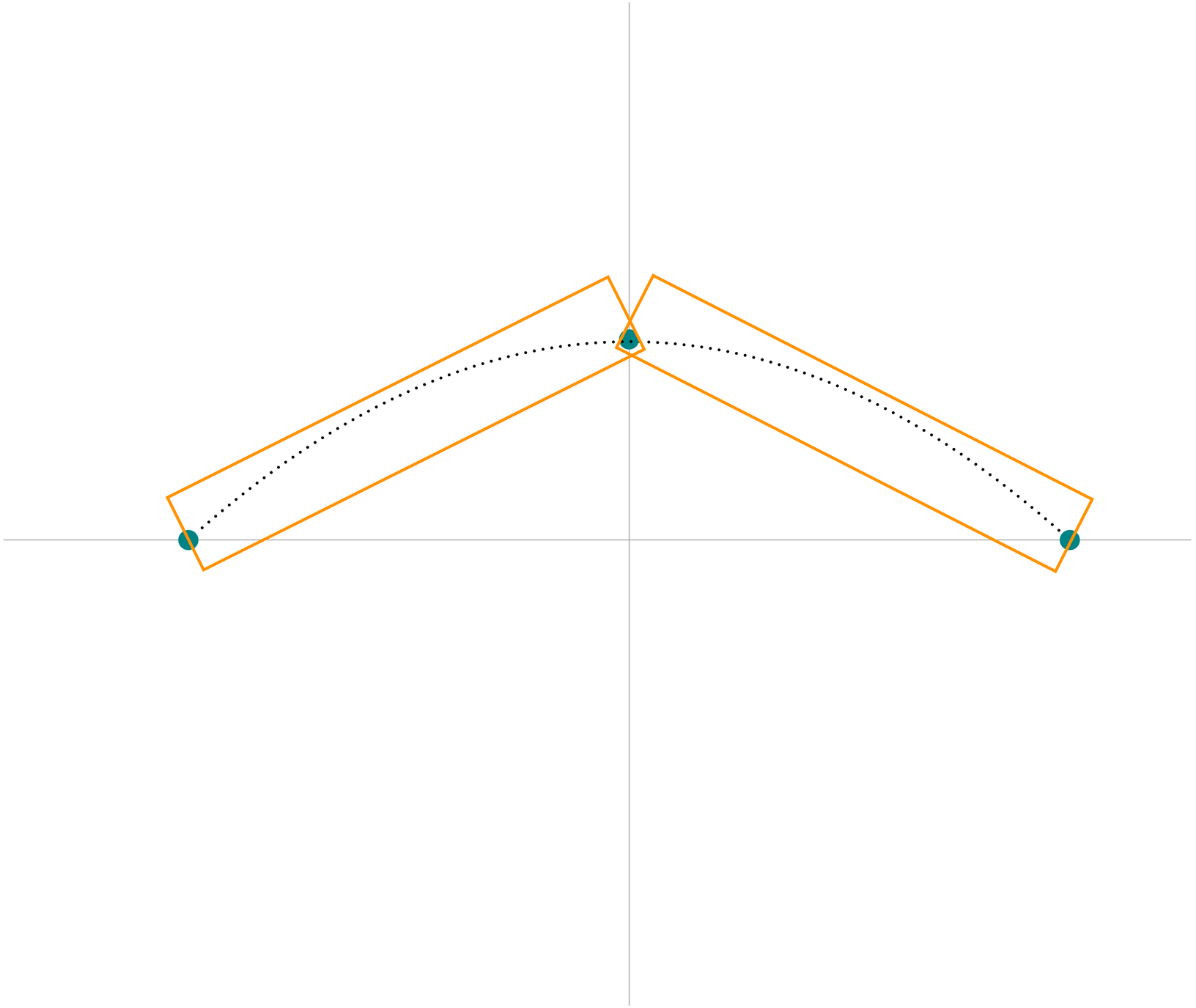
ScalaCheck

Property Based Testing









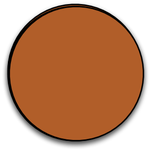
Define Success

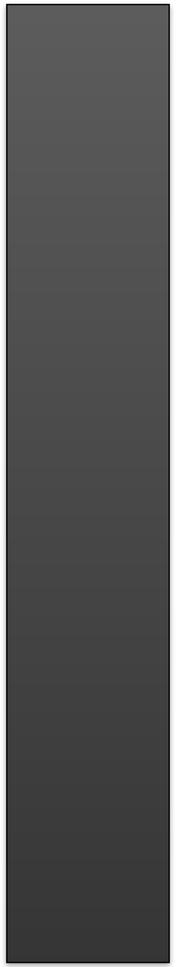
Design

Documentation

Quality

Enable Change

Generators 



Properties 

Generators

Generators

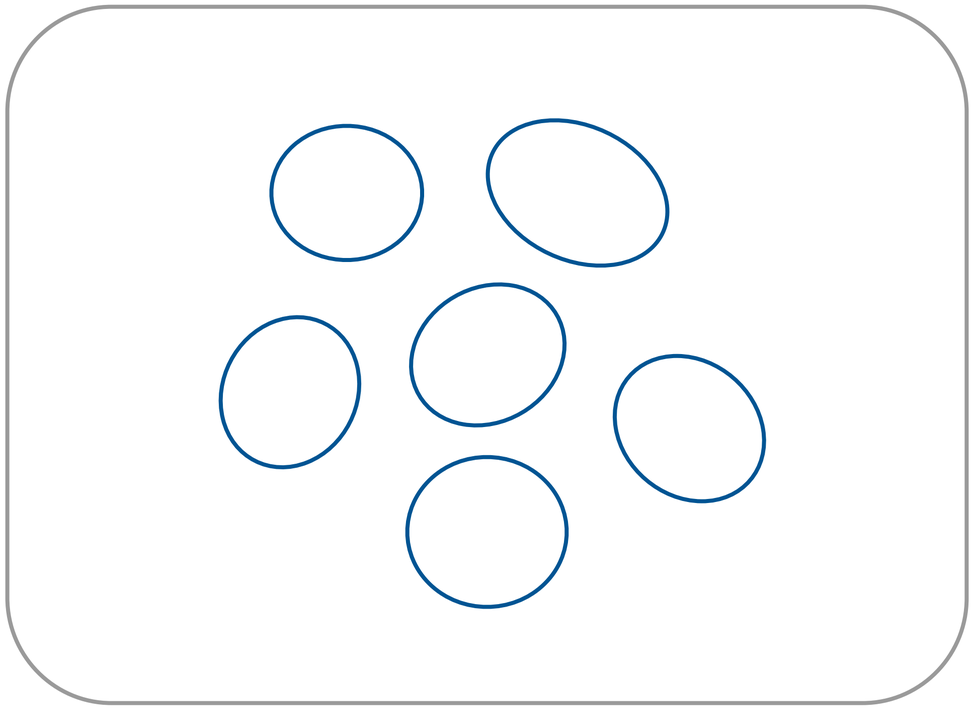
Document: what is valid input data?

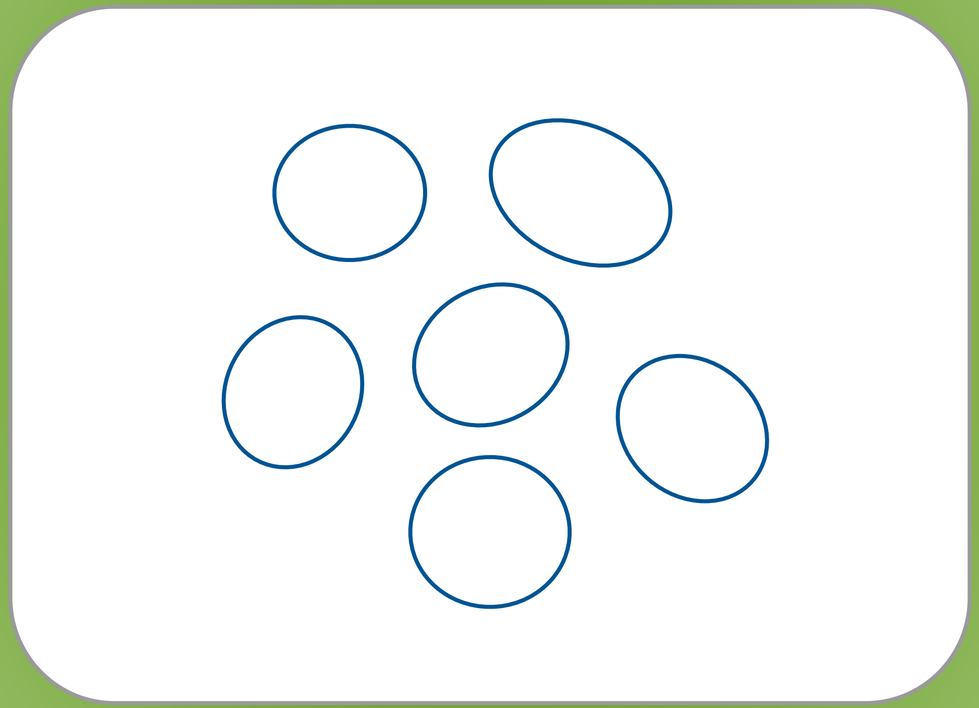
Quality: automatic corner cases

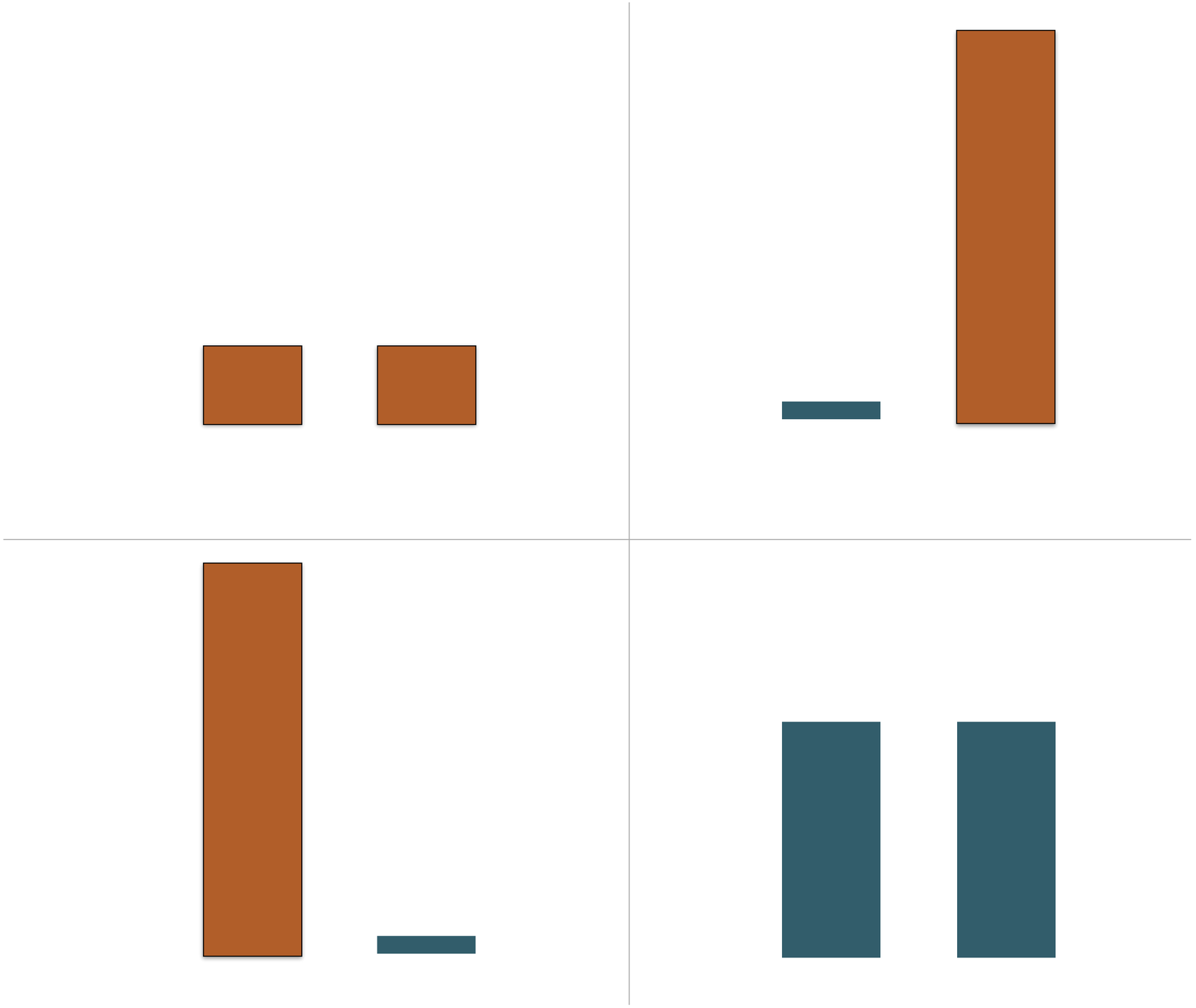
Design: build up from components

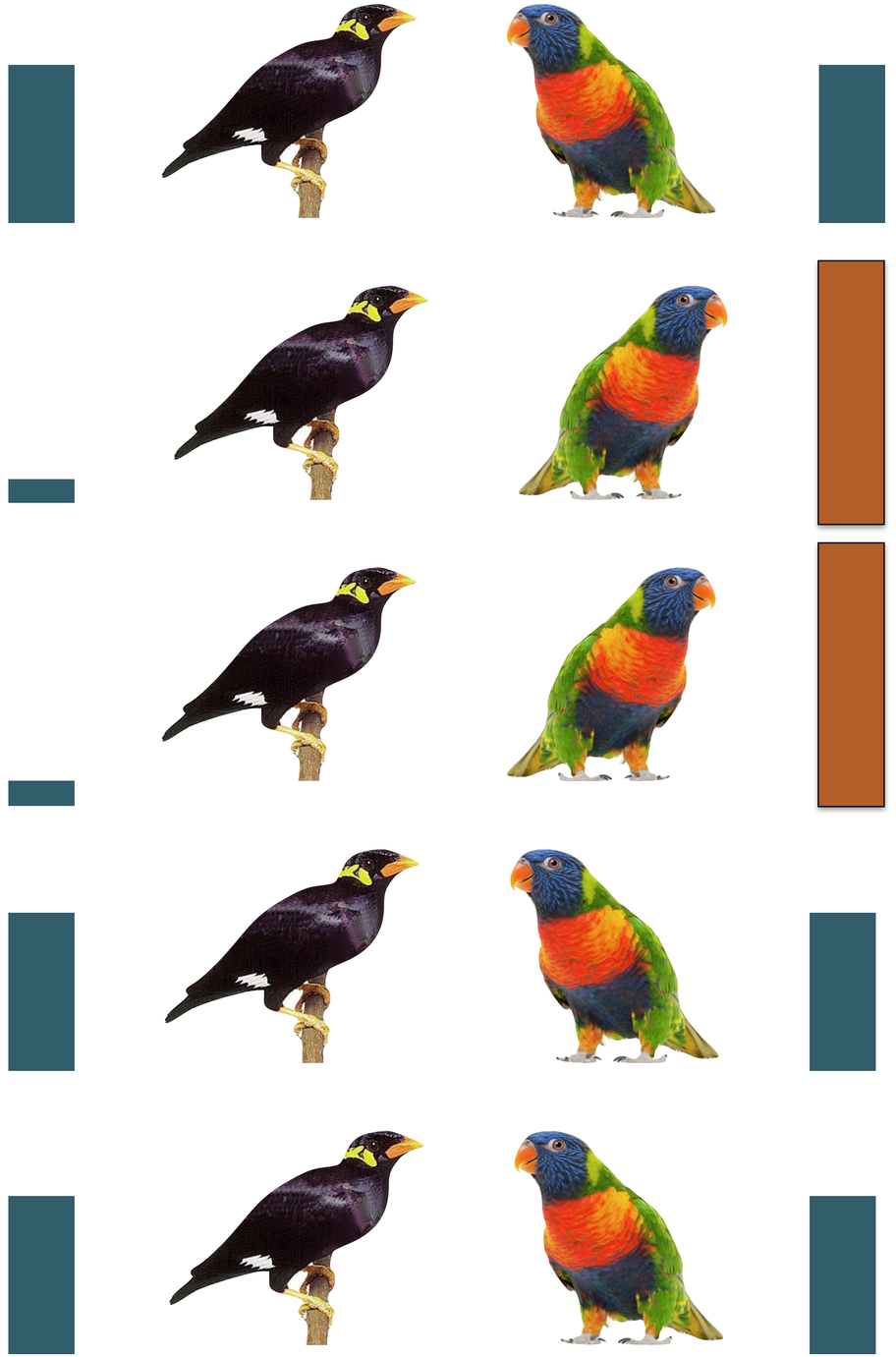
Enable: tests of larger components

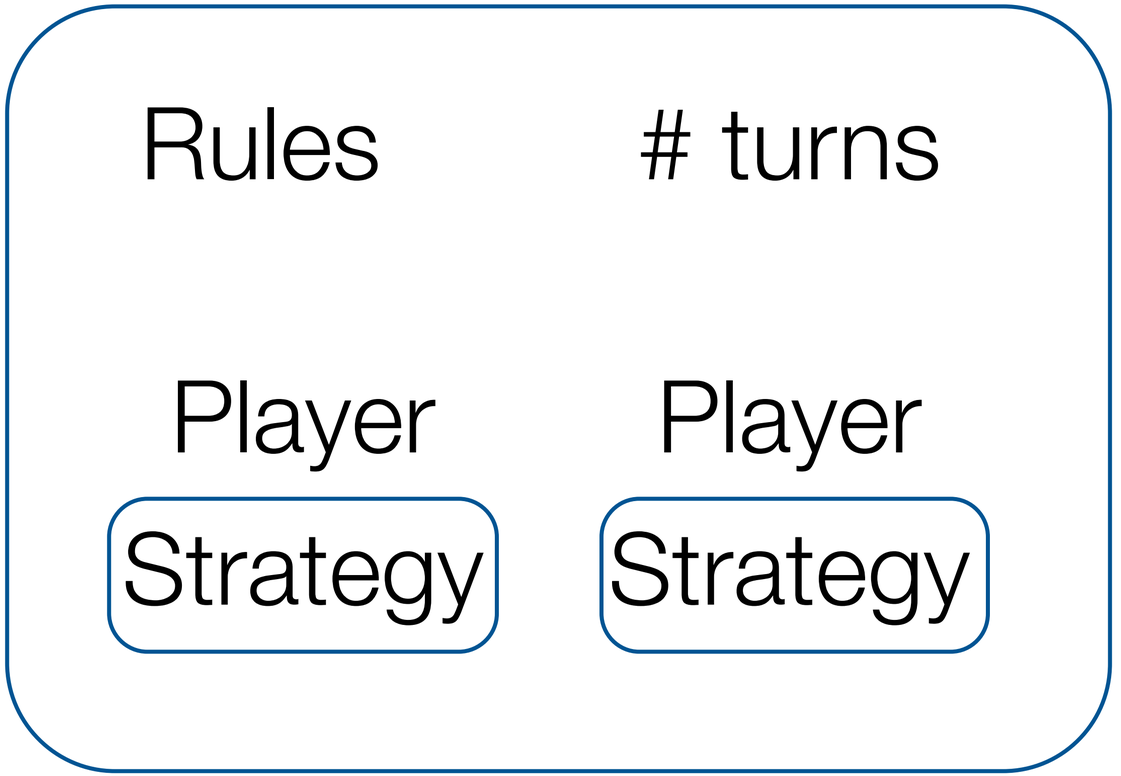
Enable: tests of larger components










Game
Rules
turns
Player
Strategy
Player
Strategy

Rules

```
case class Rules(temptationToDefect: Points,
                 rewardForMutualCooperation: Points,
                 punishmentForMutualDefection: Points,
                 suckersPenalty: Points)
```

```
object RuleTest extends Properties("Prisoners Dilemma") {


  property("The game is fair") =
    forAll {(rules: Rules, moves: MoveSet) =>
      val oneWay =       Rules.score(rules, moves)
      val theOtherWay = Rules.score(rules, moves.swap)

      oneWay.swap == theOtherWay
    }



  }
```

```
object RuleTest extends Properties("Prisoners Dilemma") {



property("Defection is always better for me") =
      forAll {(rules: Rules, theirMove: Move) =>
        val ifIDefect =
            Rules.score(rules, (Defect, theirMove))._1
        val ifICooperate =
            Rules.score(rules, (Cooperate, theirMove))._1

        ifIDefect > ifICooperate
      }


  }
```

$$A \Rightarrow B$$

Properties

Properties

Quality: they are always true

Document: what is important?

```
property("The sucker always cooperates") =
    forAll(strategyGen, Gen.posNum[Int]) {
      (opponent: Strategy, turns: Int) =>
        val allMoves:Stream[MoveSet] =
     Strategy.moves(Strategy.sucker, opponent).take(turns)

        val myMoves = allMoves.map (_._1)

        myMoves.forall(_ == Cooperate)
  }
```

Complete properties

Incomplete properties

Relational properties

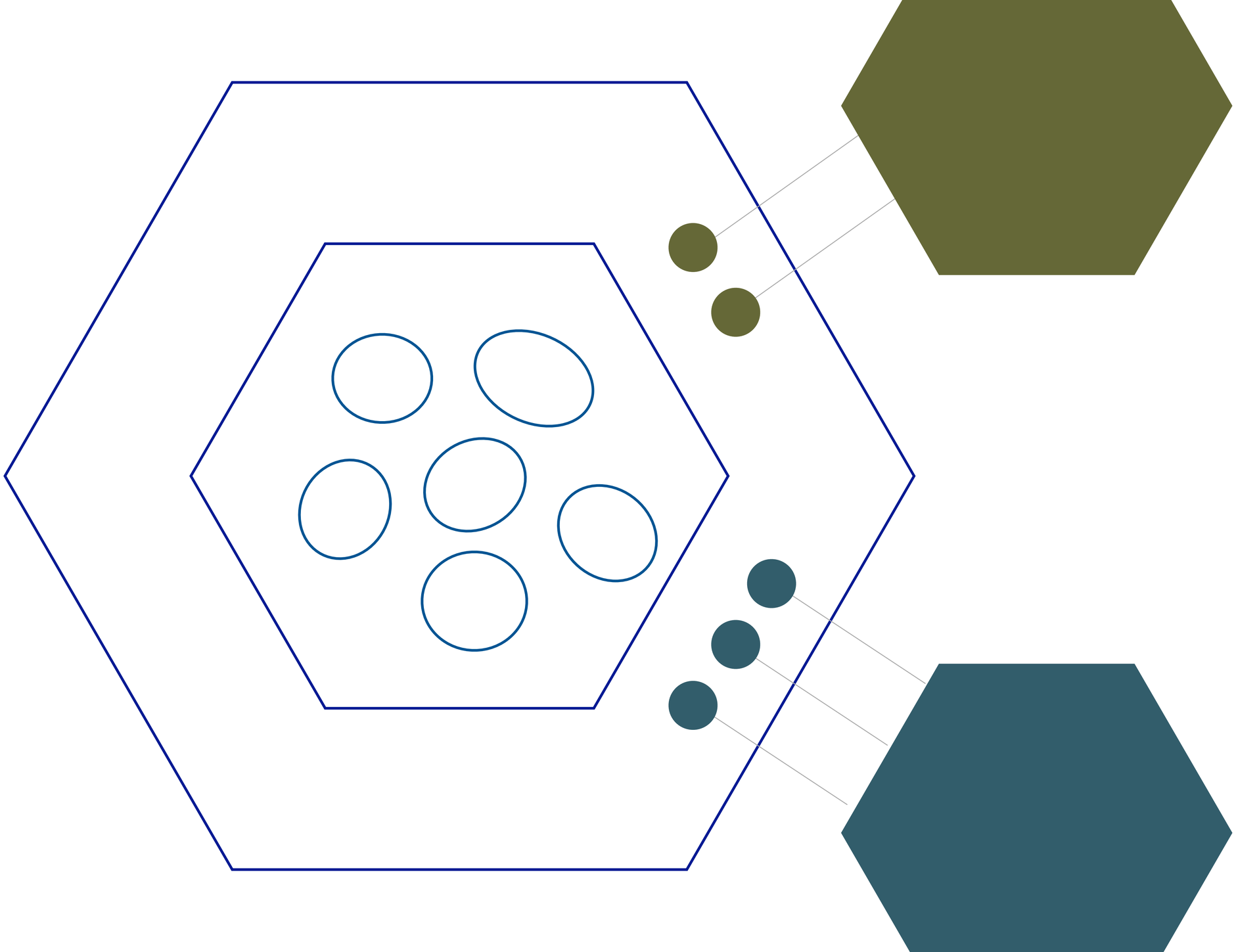
```
forAll { (list: List[Int] =>
  list.reverse.reverse = list
}
```

```
forAll { (input: Input) =>
  oldWay(input) =? newWay(input)
}
```

```
forAll { (output: Output) =>
  val input = from(output)
  subject(input) ?= output
}
```

```
property("All games end within the time limit") =
 forAll {
 (rules: Rules, players: Seq[Player], limit: Duration) =>
  classify(players.size < 10, "small", "large") {
   val timer = new Timer()
   val output = Game.eachOnEach(rules)(sys, players, limit)
   val timeTaken = timer.check
   val timeOver = timeTaken - limit

   classify (timeOver < (fudge/2), "comfortable", "barely") {
    (timeTaken <= (limit + fudge)) :|
          s"$timeTaken was longer than $limit"
  }}}
```



Properties

Quality: they are always true

Document: what is important?

Enable: changes to less important bits

Design: what do we need to know?

What does it make free?

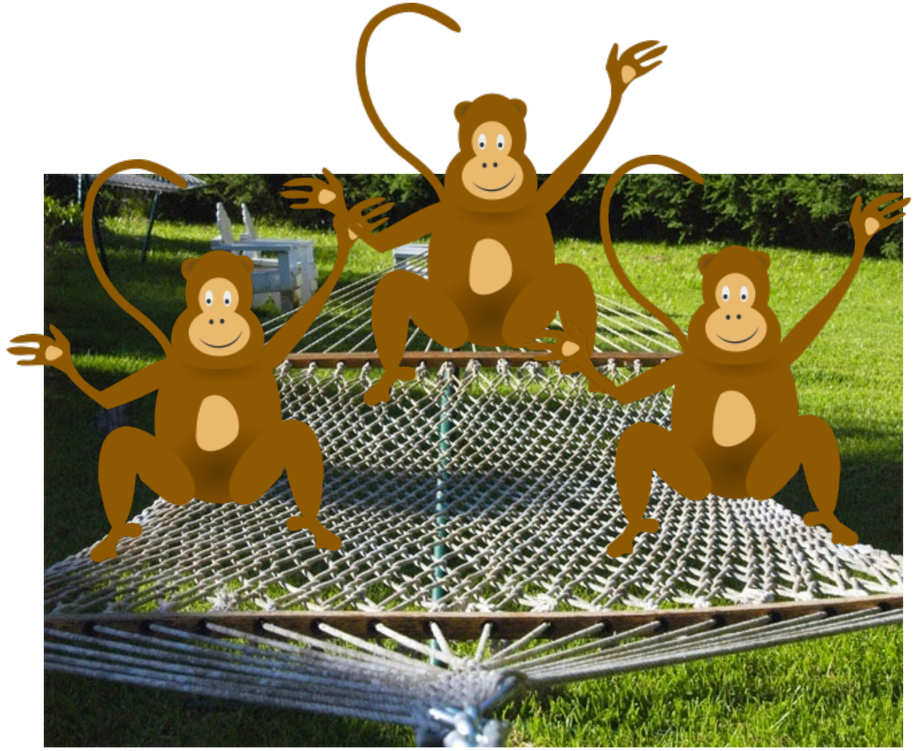
What does it make explicit?

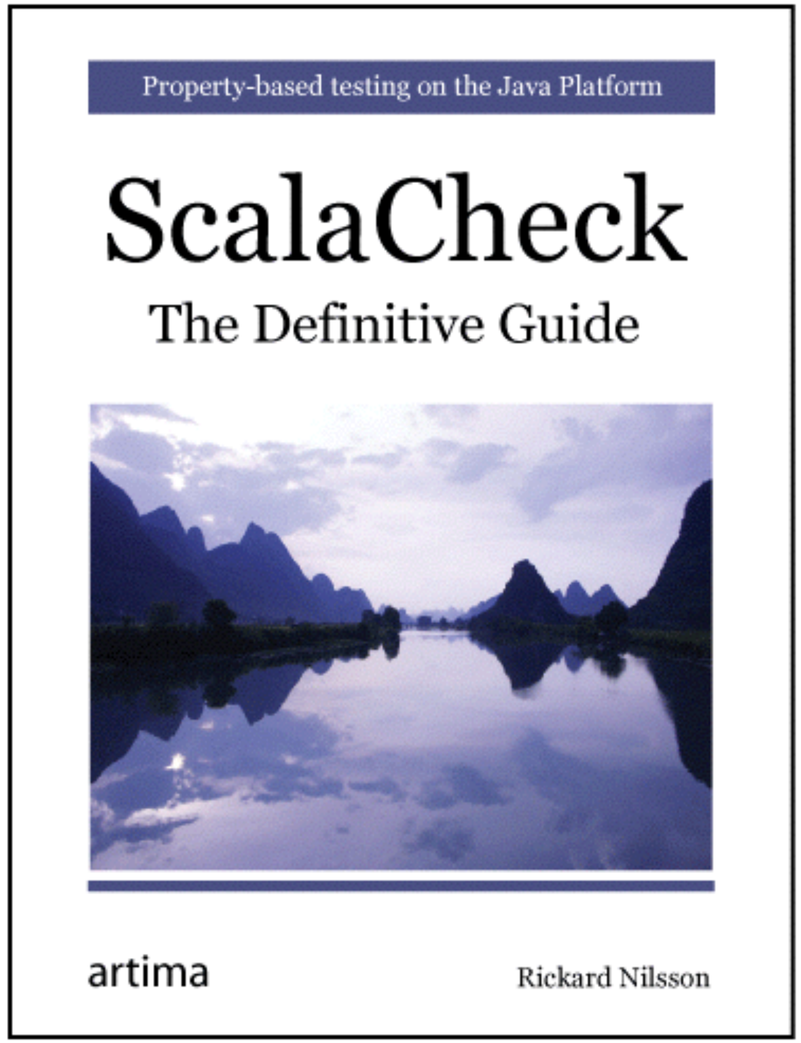
What does it make impossible?

Scala

ScalaCheck

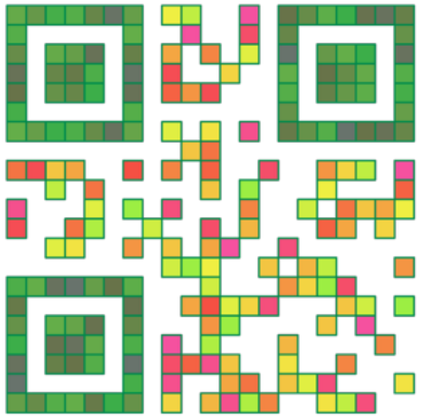
Property Based Testing





http://www.artima.com/shop/scalacheck

https://github.com/jessitron/scalacheck-prisoners-dilemma



@jessitron

Outpace™