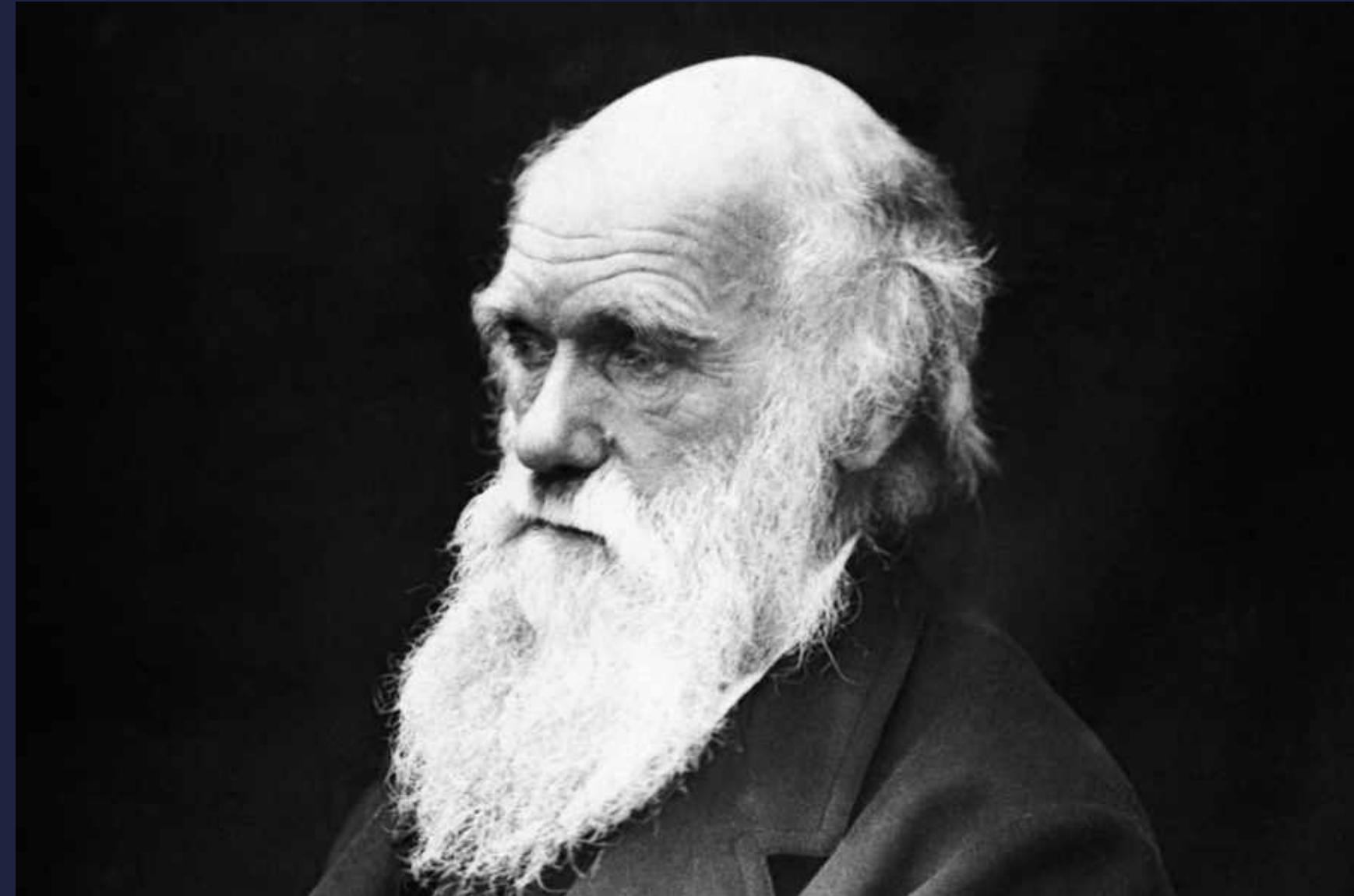


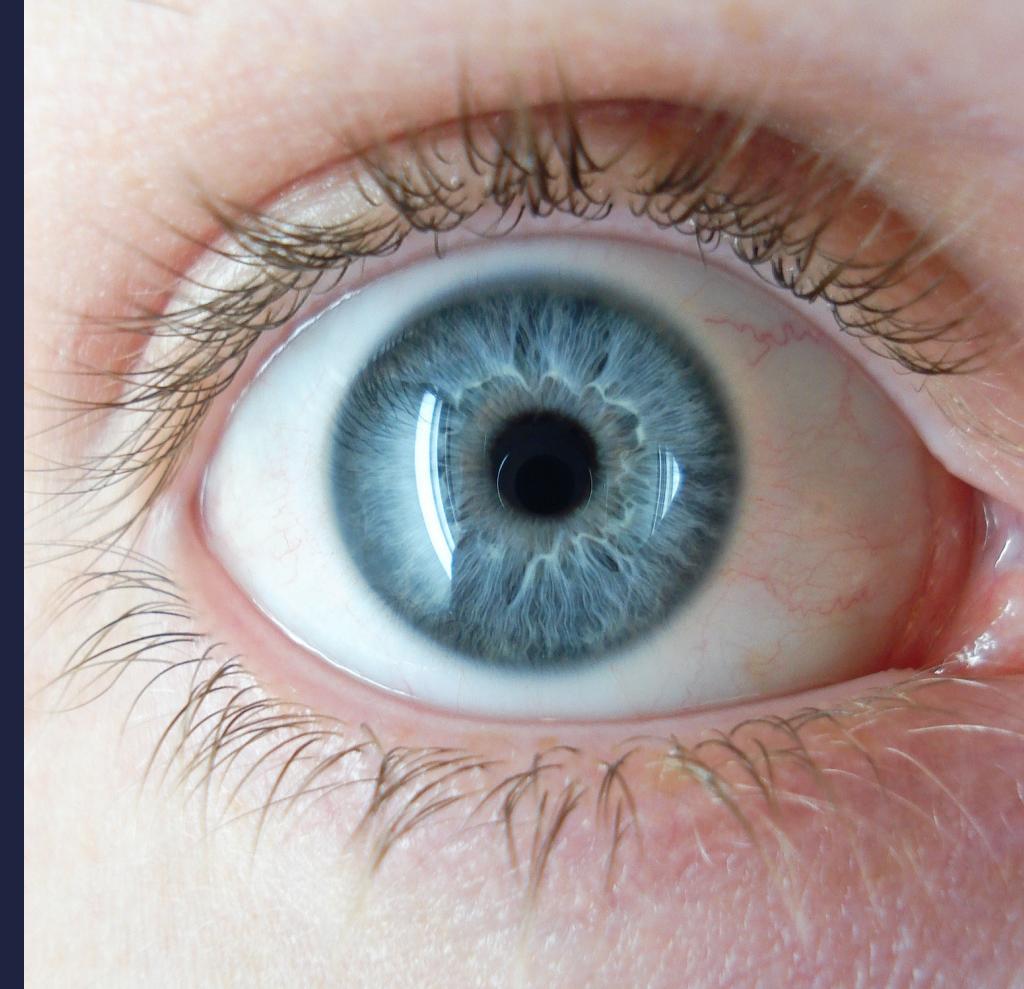
RUBIK'S CUBES AND GENETIC ALGORITHMS IN SWIFT

@JAVI

EVOLUTION BY NATURAL SELECTION



EVOLUTION BY NATURAL SELECTION





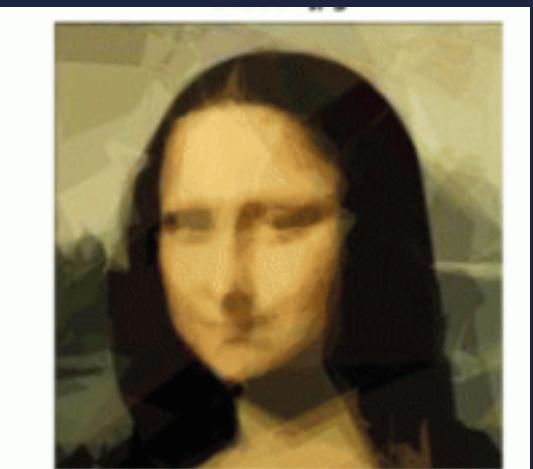
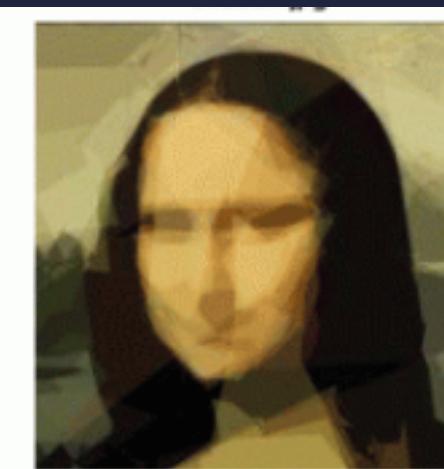
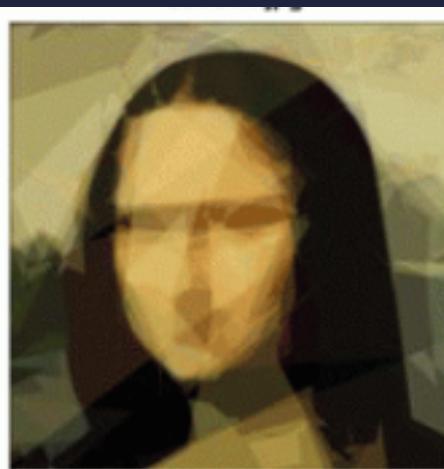
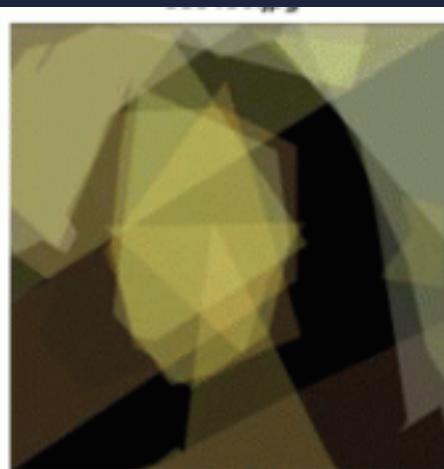
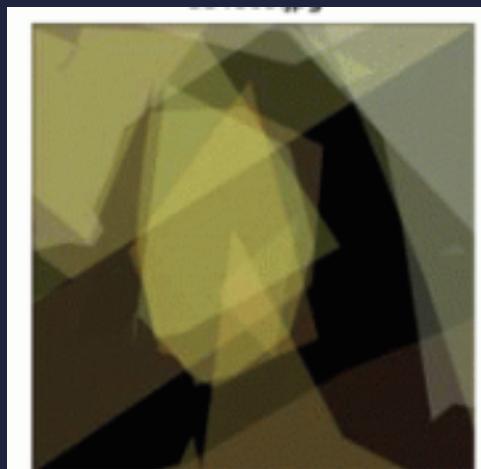
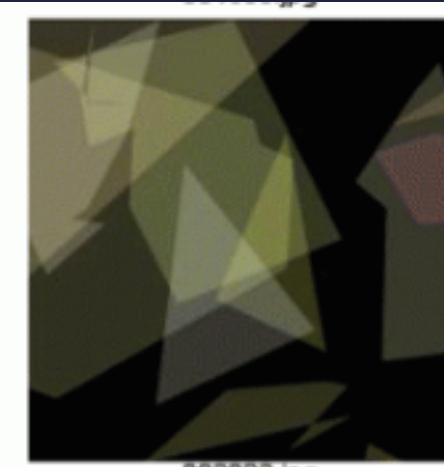
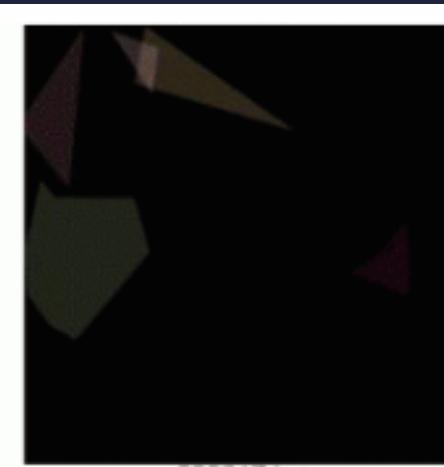
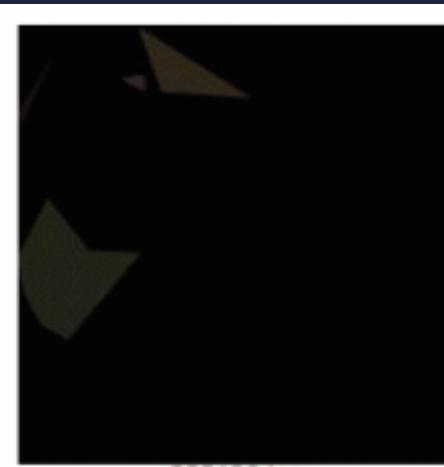
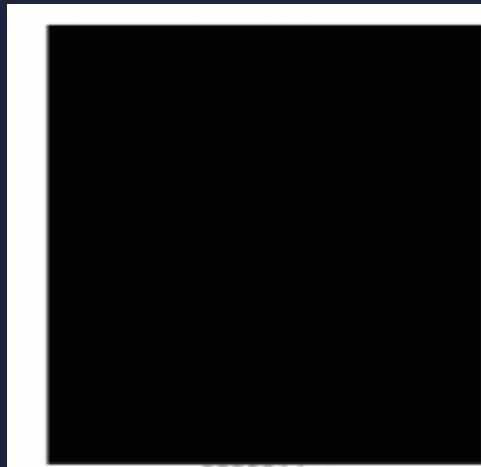
4 – "Rubik's Cubes and Genetic Algorithms In Swift" - @Javi

EVOLUTION BY NATURAL SELECTION

1. ?
2. Random mutations **in genome**
3. Survival of the fittest
4. *(Rinse and repeat)*

GENETIC ALGORITHMS

GENETIC ALGORITHMS



GENETIC ALGORITHMS - EXAMPLE

1. GOAL: FIND A NUMBER 'X'
2. START WITH 0000000000000000
3. MUTATE EACH RANDOMLY, BIT BY BIT
4. CALC FITNESS COMPARING WITH X'S BITS

GENETIC ALGORITHMS - EXAMPLE

```
final class Individual {  
    let number: UInt  
  
    init(number: UInt) {  
        self.number = number  
    }  
}
```

GENETIC ALGORITHMS - EXAMPLE

```
typealias Fitness = Int

extension Individual {
    func fitness(withGoal goal: UInt) -> Fitness {
        return Fitness(self.number.numberOfBitsEqual(in: goal))
    }
}
```

GENETIC ALGORITHMS - EXAMPLE

```
extension Individual {  
    let likelihoodOfMutatingBit = 5  
  
    func mutate() -> Individual {  
        let mutatedBits = self.number.bits.map { bit -> Bool in  
            let shouldMutate = Bool.random(percentage: likelihoodOfMutatingBit)  
  
            return shouldMutate ? !bit : bit  
        }  
  
        return Individual(number: UInt(bits: mutatedBits))  
    }  
}
```

GENETIC ALGORITHMS - EXAMPLE

```
final class GeneticSolver {
    var population: [Individual]

    init(numberToSolve: UInt, populationSize: Int)

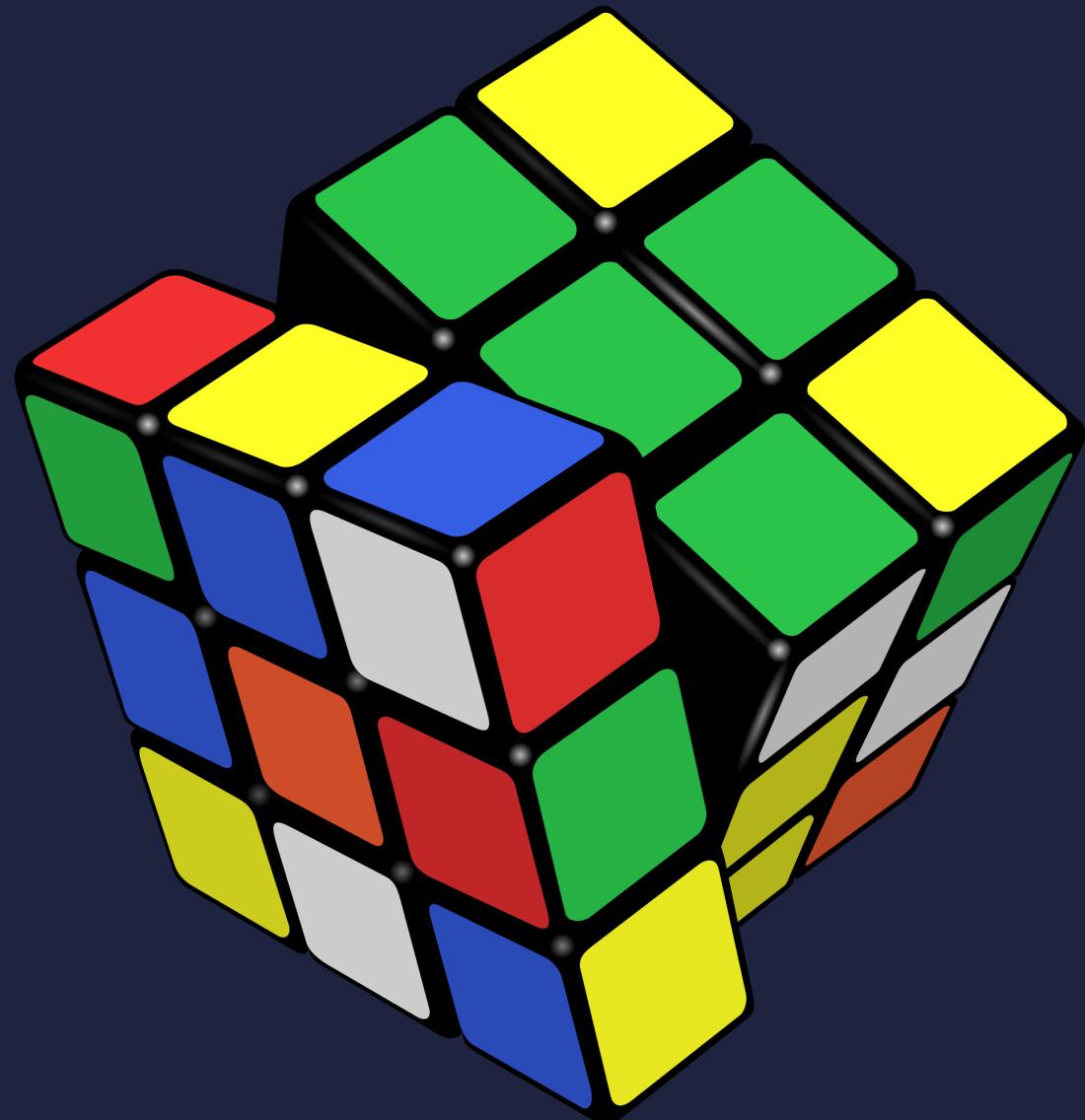
    func runGeneration() {
        // 1: Natural selection (survival of the fittest)
        population.removeLast(Int(Double(population.count) * 0.8))

        // 2: Random mutations
        population += population.map { $0.mutate() }

        // 3: Sort by fitness
        population.sort { $0.fitness(towards: numberToSolve) > $1.fitness(towards: numberToSolve) }
    }
}
```

GENETIC ALGORITHMS - EXAMPLE

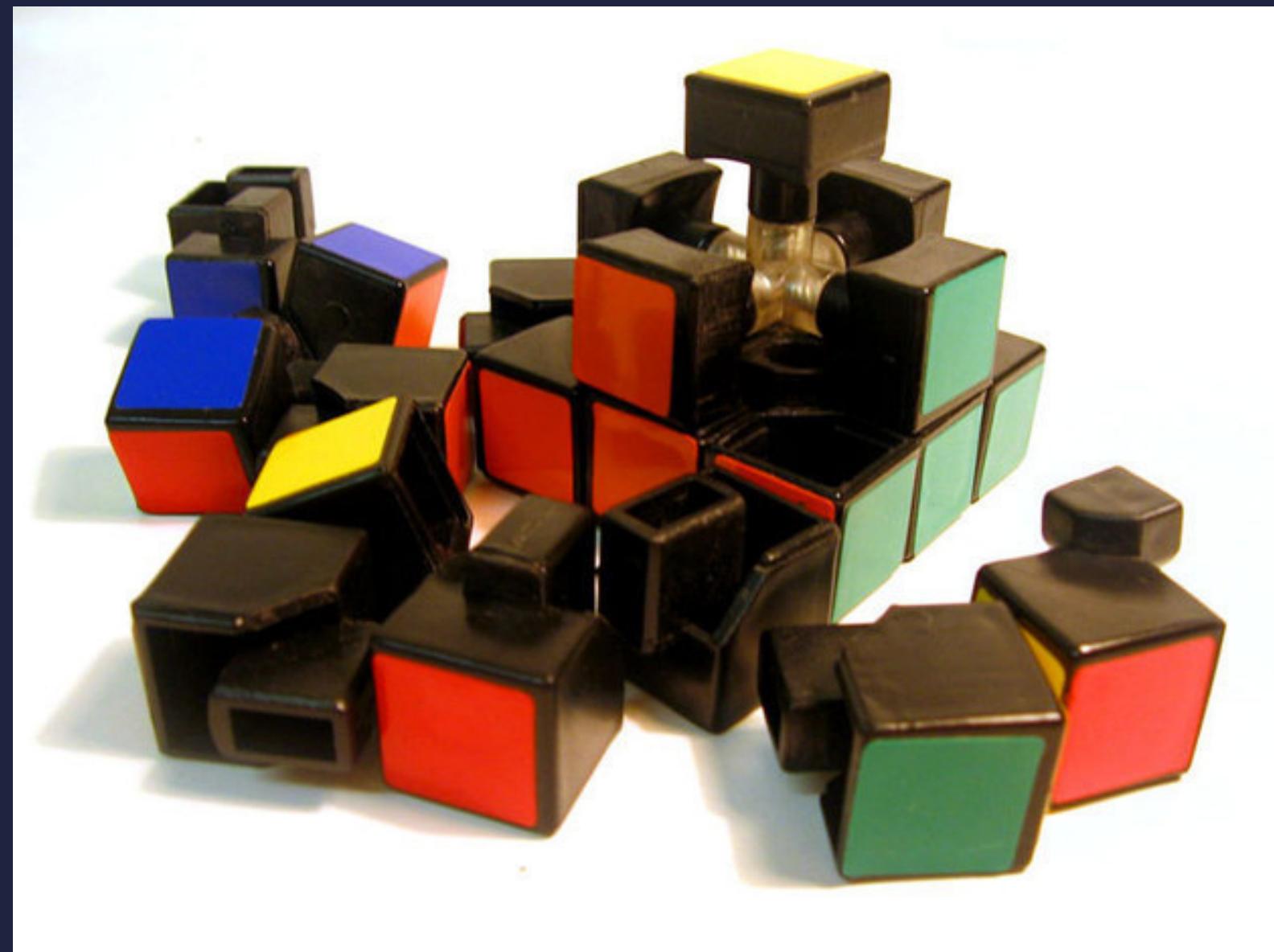
RUBIK'S CUBES



RUBIK'S CUBES



RUBIK'S CUBES



RUBIK'S CUBES

43,252,003,274,489,856,000

RUBIK'S CUBES



RubikSwift

```
struct Cube {  
    struct Pieces {  
        var edges: EdgePieceCollection  
        var corners: CornerPieceCollection  
    }  
  
    var pieces: Pieces  
}
```

RubikSwift

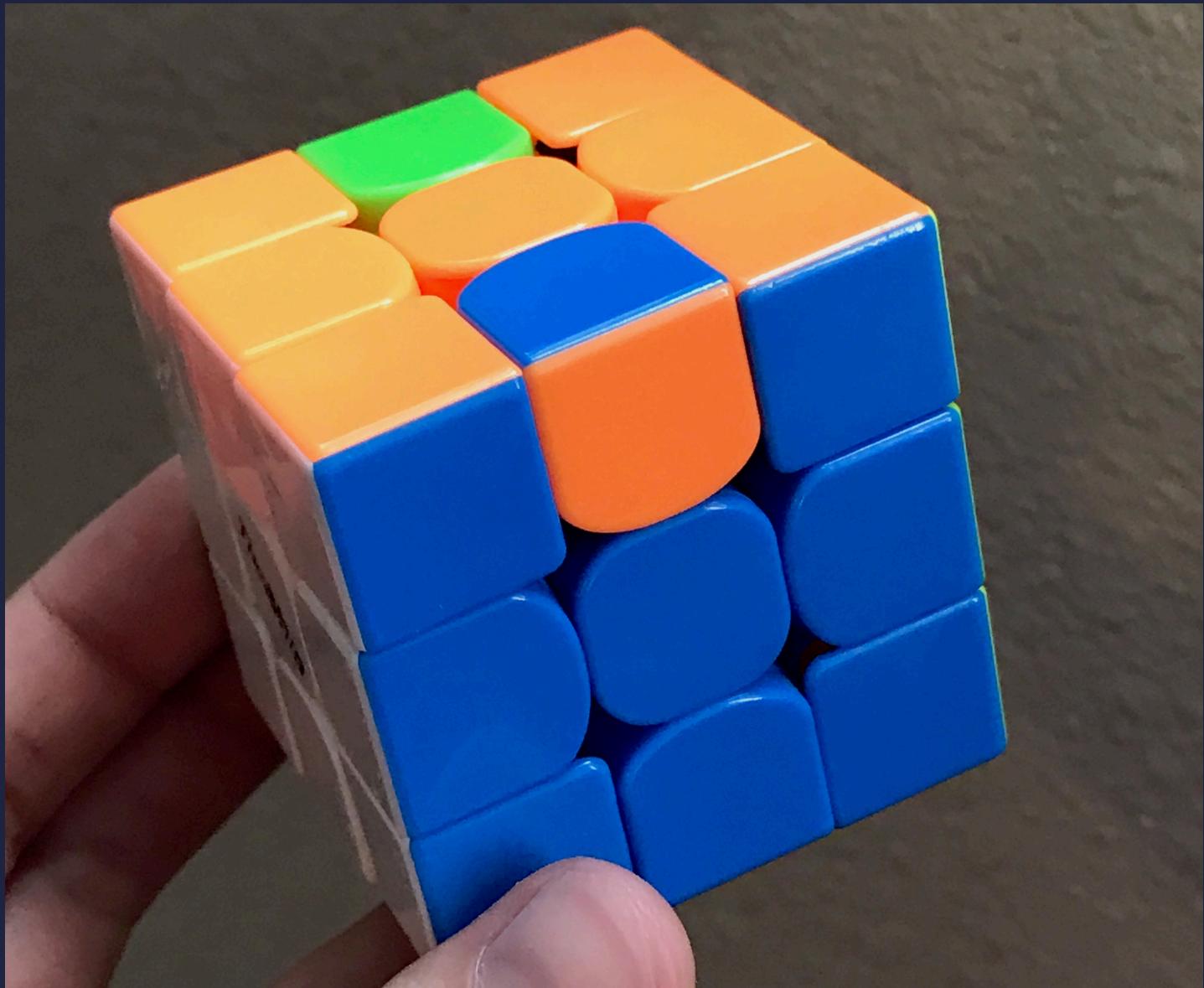
```
enum EdgeLocation {  
    case topRight, topFront, topLeft, topBack...  
}
```

```
struct EdgePiece {  
    enum Orientation {  
        case correct  
        case flipped  
    }  
}
```

```
    var location: EdgeLocation  
    var orientation: Orientation
```

```
}
```

RubikSwift



RubikSwift

```
struct CornerPiece {  
    enum Orientation {  
        case correct  
        case rotatedClockwise  
        case rotatedCounterClockwise  
    }  
  
    var location: CornerLocation  
    var orientation: Orientation  
}
```

RubikSwift

```
struct Cube {  
    struct Pieces {  
        var edges: EdgePieceCollection  
        var corners: CornerPieceCollection  
    }  
  
    var pieces: Pieces  
}
```

RubikSwift

```
enum Face {  
    case top, bottom, left, right, front, back  
}  
  
struct Move {  
    enum Magnitude {  
        case clockwiseQuarterTurn  
        case halfTurn  
        case counterClockwiseQuarterTurn  
    }  
  
    let face: Face  
    let magnitude: Magnitude  
}
```

RubikSwift



RubikSwift

```
extension Cube {  
    mutating func apply(_ move: Move)  
}
```

RubikSwift

```
final class Individual {  
    let moves: [Move]  
  
    init(moves: [Move]) {  
        self.moves = moves  
    }  
}
```

RubikSwift

```
typealias Fitness = Int

extension Individual {
    func fitness(solvingCube cube: Cube) -> Fitness {
        var cubeAfterApplyingMoves = cube
        cubeAfterApplyingMoves.apply(self.moves)

        return cubeAfterApplyingMoves.numberOfSolvedPieces
    }
}

extension Cube {
    // Number of pieces in the correct location and orientation
    var numberOfSolvedPieces: Int
}
```

RubikSwift

```
extension Individual {  
    func mutate() -> Individual {  
        var moves = self.moves  
  
        let randomMoves = Move.randomMoves(count: Int.random(in: 5...20))  
  
        moves += randomMoves  
  
        return Individual(moves: moves)  
    }  
}
```

RubikSwift

```
final class Solver {
    private(set) var population: [Individual]

    init(scrambledCube: Cube, populationSize: Int)

    func runGeneration() {
        population.removeLast(Int(Double(population.count) * 0.8))

        population += population.map { $0.mutate() }

        population.sort {
            $0.fitness(solvingCube: scrambledCube) >
            $1.fitness(solvingCube: scrambledCube)
        }
    }
}
```

Rubik Swift DOES IT ACTUALLY WORK?

RubikSwift

Cube: R' F' B D2 L' R' D2 R' F2 B2 D F L' R2 F' U2 R' U' D F

1: New best: B2 (3.0)

3: New best: B2 U D' U2 F2 D2 B U2 D R B' [...] (6)

7: New best: B2 L2 U2 L' B2 F' U' U' D L2 [...] (7)

100 (avg fitness 3.6012, 23806 algs/sec, 0 min elapsed)

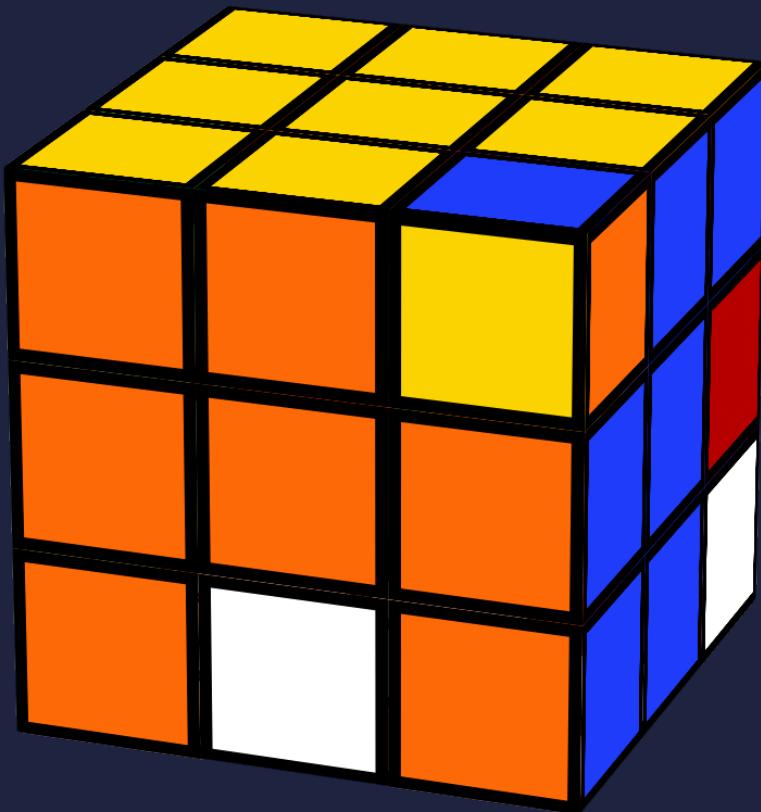
3465: New best: B2 D2 U R2 B2 U2 B' F2 D [...] (13)

10000 (avg fitness 6.462, 11217 algs/sec, 74 min elapsed)

25781: New best: L' D' B2 L2 B' D R2 F R' [...] (**16**)

80800 (avg fitness 9.2612, 7676 algs/sec, 877 min elapsed)

RubikSwift



Fitness: 16

LINKS

- <http://github.com/JaviSoto/talks>
- <http://github.com/JaviSoto/RubikSwift>
- [https://rogeralsing.com/2008/12/07/genetic-programming-
evolution-of-mona-lisa/](https://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/)

@JAVI

Thank you

