JugaMe

experimentando con juegos

Teoría de Juegos

jugar no es (sólo) cosa de chicos

Teoría de Juegos

Área de la matemática aplicada que utiliza modelos para estudiar interacciones en estructuras formalizadas de incentivos y llevar a cabo procesos de decisión. Estudia las estrategias óptimas, así como el comportamiento previsto y observado de los individuos.

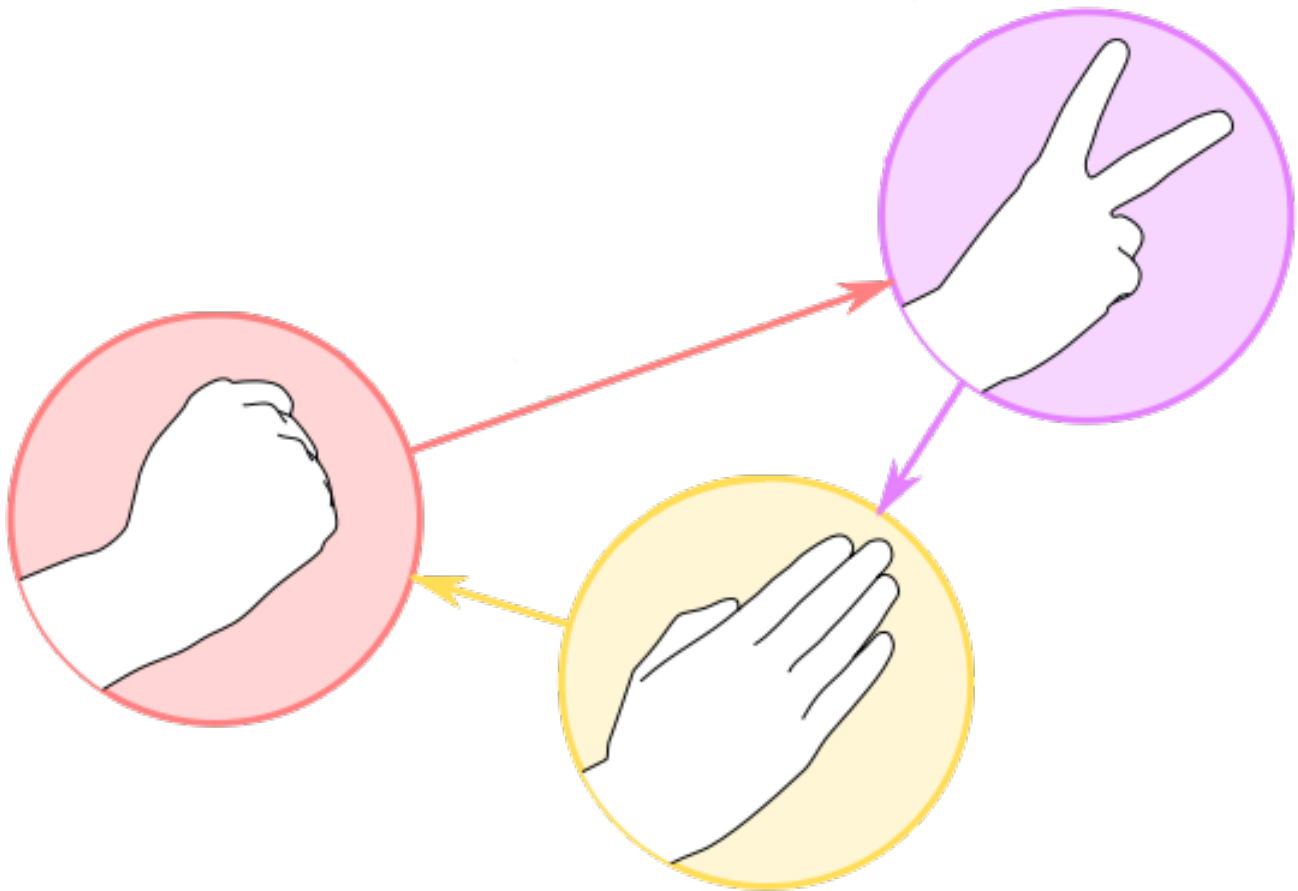
"En entornos complejos, los individuos no son totalmente capaces de analizar la situación y calcular la estrategia óptima" (Nash)

Juegos a analizar

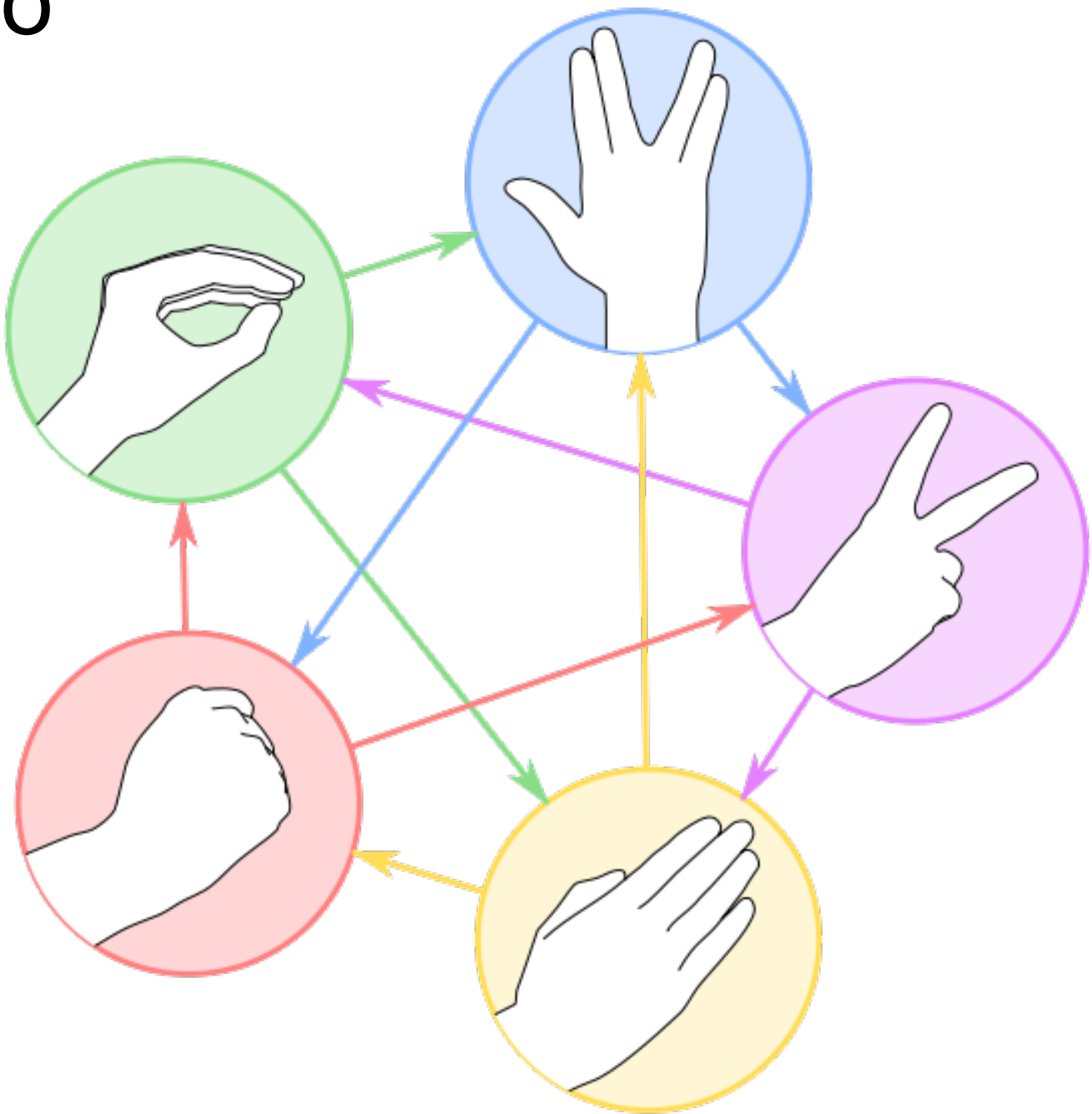
Características:

- El *Objetivo* genera *Conflicto*
- Existen reglas
- El resultado depende de las decisiones de todos
- Hay distintos tipos: Suma cero/no cero, con información completa/incompleta, cooperativos/no cooperativos

Ejemplo



Ejemplo



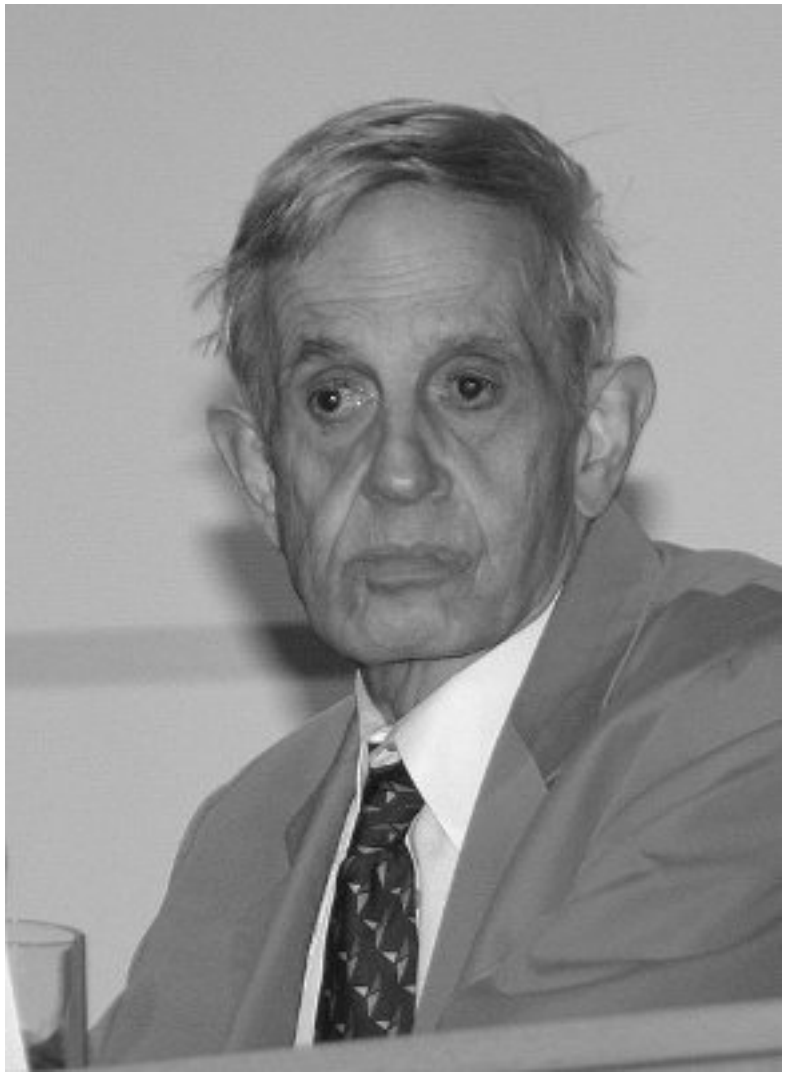
Funcionamiento de un juego

- Existe una matriz de pagos
- Cada uno toma su decisión
- Se calculan los pagos

	Cooperar	**Desertar**
Cooperar	3, 3	-5, 5
Desertar	5, -5	-1, -1

Si se juega en modo "reiterado" se espera que cada uno utilice la estrategia que le permite obtener el mayor beneficio. En este caso, las decisiones previas pueden afectar las futuras

Grandes Jugadores



J. Nash

R. Axelrod

DSLs

hablando un mismo idioma

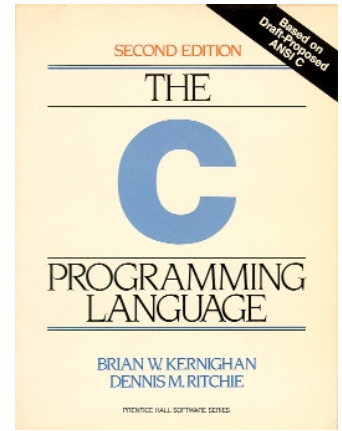
DSLs

"The basic idea of a domain specific language (DSL) is a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem."

Martin Fowler

DSLs y GPLs

SQL



^regex$

Internos vs Externos

by Martin Fowler

Internos:

- Hay un lenguaje host flexible
- Se lo usa de una manera limitada y particular

Externos:

- Se usa un lenguaje que no es el de la aplicación principal:
  - Ad-hoc
  - Sobre otros lenguajes (XML, por ejemplo)

Internos ¿vs? Externos

Crítica común:

"Si las abstracciones siguen siendo clases y métodos, es una API y no un DSL"

Contracrítica:

"*Una API que permite hablar en términos del negocio, es un DSL* "

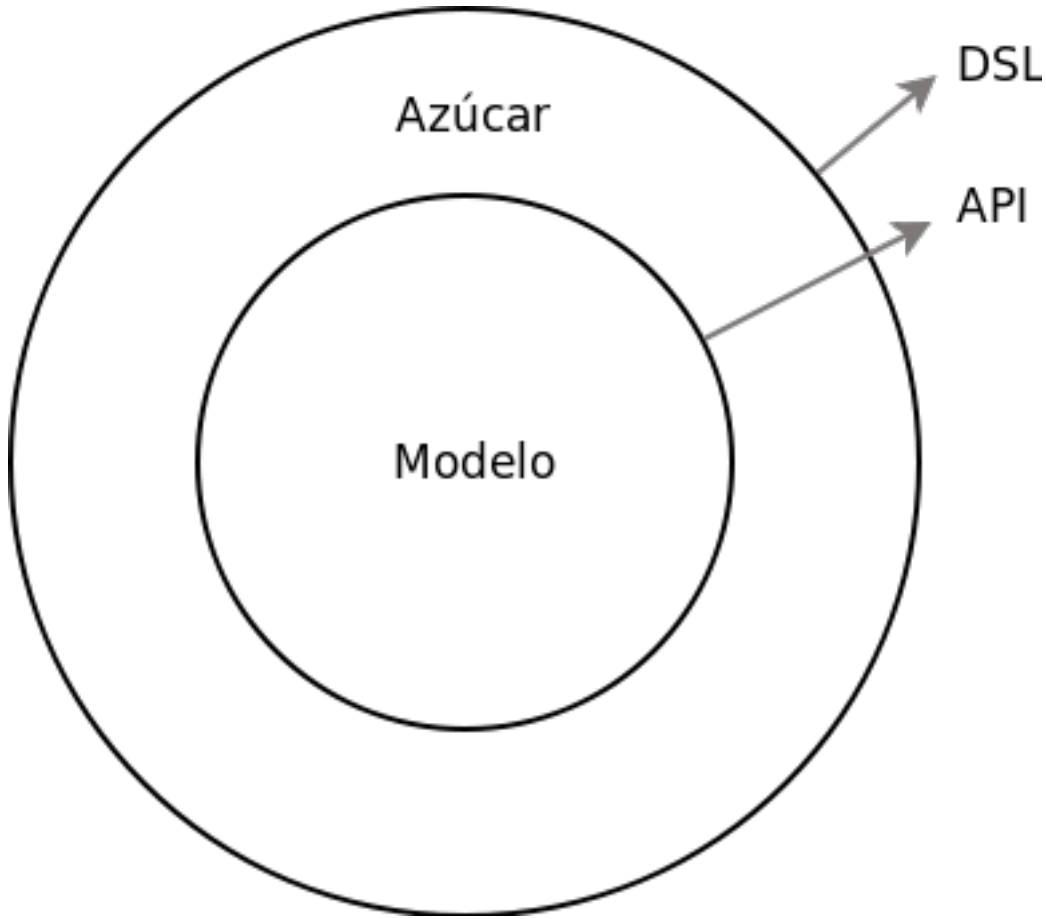
"*Fluent Interfaces*"

Internos ¿vs? Externos

```
TimePoint fiveOClock, sixOClock;
...
TimeInterval meetingTime = new TimeInterval(fiveOClock,
sixOClock);
TimeInterval fluentMeetingTime = fiveOClock.until(sixOClock);
```

"*Fluent Interfaces*"

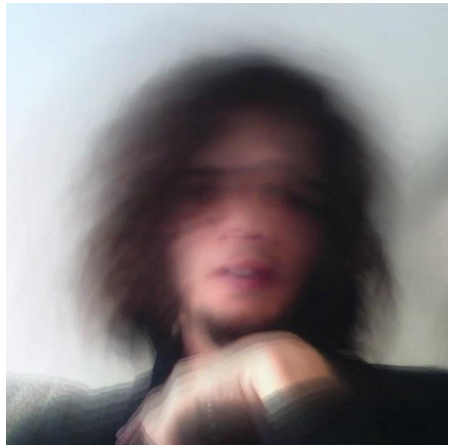
Internos



Jugame


+

+

+


=

Objetivos

DSL +  + Juegos

RSpec

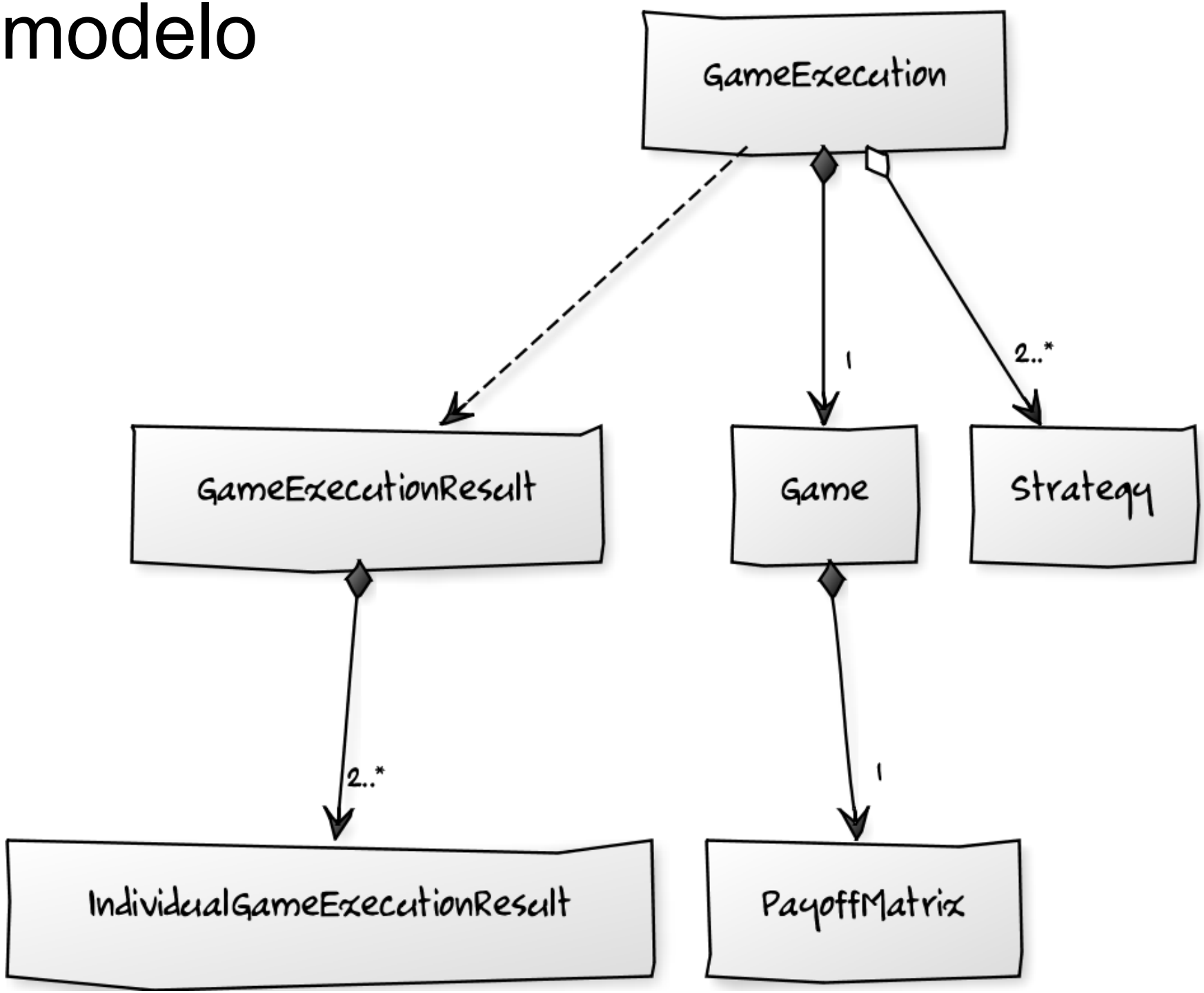
La implementación

a jugar

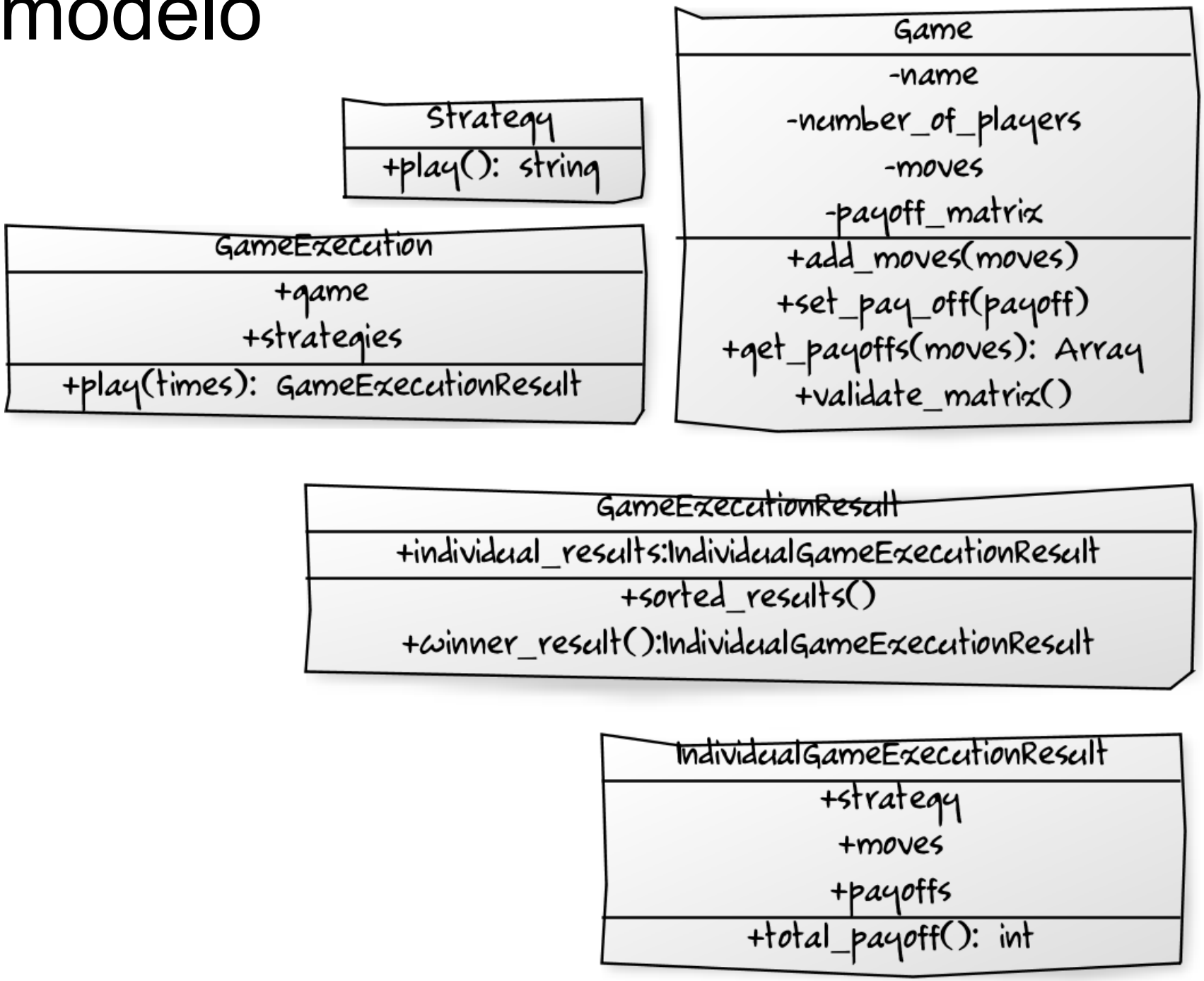
Ruby

- Lenguaje de Propósito General
- Inspirado en Perl, Smalltalk, Eiffel, Ada, Lisp, etc.
- Todo es un objeto (en serio)
- Abierto y Libre (as in Freedom, as in Free beer, as in Open Source, as in ...)
- Con Mixin múltiple ;)
- Duck Typing

El modelo


GameExecution
GameExecutionResult
Game
Strategy
IndividualGameExecutionResult
PayoffMatrix
1
2..*
2..*
1

El modelo


Strategy
+play(): string
Game
-name
-number_of_players
-moves
-payoff_matrix
+add_moves(moves)
+set_pay_off(payoff)
+get_payoffs(moves): Array
+validate_matrix()
GameExecution
+game
+strategies
+play(times): GameExecutionResult
GameExecutionResult
+individual_results:IndividualGameExecutionResult
+sorted_results()
+winner_result():IndividualGameExecutionResult
IndividualGameExecutionResult
+strategy
+moves
+payoffs
+total_payoff(): int

A la antigua

```
context "game being initialized" do
  it "should return values according to its payoff matrix (2 options, 2 players)" do
    @game = JugaMe::Game.new("The Prissioner's Dilemma",2)
    @game.add_moves(["Stay Silent", "Betray"])
    @game.set_pay_off ["Stay Silent", "Stay Silent"], [1, 1]
    @game.set_pay_off ["Stay Silent", "Betray"], [10, 0]
    @game.set_pay_off ["Betray", "Stay Silent"], [0, 10]
    @game.set_pay_off ["Betray", "Betray"], [5, 5]

    @game.get_pay_off(["Stay Silent", "Stay Silent"]).should eql [1,1]
    @game.get_pay_off(["Stay Silent", "Betray"]).should eql [10,0]
    @game.get_pay_off(["Betray", "Stay Silent"]).should eql [0,10]
    @game.get_pay_off(["Betray", "Betray"]).should eql [5,5]
  end
end
```

¡JugaMe!

```
context "game being created" do
  it "should have a name, players and payoff matrix (2 players)" do

    in_the_game "The Prissioner's Dilemma", 2.players do
      choose_to "Betray"
      choose_to "Stay Silent"
      when_their_choices_are "Betray", "Betray", they_pay = 5
      when_their_choices_are "Stay Silent", "Stay Silent", they_pay = 1
      when_their_choices_are "Stay Silent", "Betray", they_pay = 10, 0
      when_their_choices_are "Betray", "Stay Silent", they_pay = 0, 10
    end

    game = JugaMe::Game.games["The Prissioner's Dilemma"]
    game.should_not be_nil
    game.name.should eql "The Prissioner's Dilemma"
    game.number_of_players.should be 2
    game.moves.length.should be 2
  end
end
```

Implementación

POROs que implementan la lógica de juegos

= ----------

API

\+ Builders

= ----------

Fluent Interface

\+ Instancias estáticas

\+ Acceso "natural" a los builders en Object

DSL

Referencias

- Martin Fowler:
  - "Domain Specific Language"
  - "Introduction to Domain Specific Languages"

- Robert Axelrod:
  - "Evolving New Strategies - The Evolution of Strategies in the Iterated Prissioner's Dilemma"
  - "Tit-for-tat Strategies"
  - "On Six Advances in Cooperation Theory"



Preguntas, Ideas, Críticas...


Gracias