Has-k-Dice

Sebastián Zimmermann

Enero 2020

1 Idea general

1.1 Inspiración

Este proyecto nace como inspiración del proyecto Troll que es un DSL para simular tiradas de dados, y por lo tanto, los objetivos son similares. Mi razón personal para elegir el proyecto es porque, en un primer lugar, me parecía un desafío divertido y en segundo lugar, me sería útil tener un programa que haga eso en partidas o simulaciones.

El lenguaje Troll está armado en ML (específicamente en MoscowML) por lo que me pareció interesante intentarlo en Haskell. El código no es abierto, y en parte algo que sumaría tener un proyecto de este estilo es hacerlo Open Source.

Troll es un lenguaje que lleva al menos 10 años de actualización (con períodos espaciados de tiempo), y por lo tanto es bastante grande. Me cuesta marcar una línea de hasta dónde hacer y no, por lo que **propondré distintas alternativas**

1.2 Objetivos del DSL

¿Por qué hacer un DSL para tiradas de dados?

- Descripciones concisas y no ambiguas que favorecen a la comunicación
- Servidores de internet de dados
- Calcular tus probabilidades (como jugador)
- Decidir la dificultad de algo (como GM)
- Tener un lenguaje que automatiza tiradas de dados para cualquier juego o sistema que lo necesite

2 Notación

En diversos juegos de rol, se utiliza cierta notación para representar tiradas de dados. Por ejemplo $3d6 + 4$ representa 3 tiradas de un dado de 6 caras y luego sumarle 4 al resultado.

Sin embargo hay muchos tipos de tiradas de dados que *no pueden ser representadas mediante esta notación*. Bajo esta idea, se han empleado lenguajes que atentan a crear una notación universal, entre estos **Roll**, que es antecesor de **Troll**, en el cual también basaré la notación del lenguaje HaskDice

2.1 Aclaraciones matemáticas

Una tirada de dados es una colección (multiset) de números donde:

- No importa el orden
- La cantidad de ocurrencias es importante.

Una colección de un elemento puede ser usada como un número. Las colecciones pueden ser combinadas, filtradas, contadas o sumadas para alcanzar un resultado final.

3 Semánticas y Operaciones

Troll tiene dos semánticas distintas, una para tiradas aleatorias y otra para cálculos de distribuciones de probabilidad. En este trabajo planeo mínimamente hacer gran parte de la semántica de tiradas aleatorias.

3.1 Sobre Colecciones

Al ser conjuntos, las colecciones tienen acceso a todas las operaciones de conjuntos, como unión, intersección, etc, sin embargo estas operaciones no están implementadas para los usuarios del lenguaje, ya que pueden emularse con las operaciones ya implementadas y resulta poco probable la necesidad de realizar una intersección o una diferencia entre tiradas de dados. La excepción de esto es la unión, que está "implementada" (planeo utilizar el Data.Multiset provisto por Haskell) y se representa con el caracter **@**.

3.2 Operaciones básicas de tiradas aleatorias

- dN hace una sola tirada de un dado de N caras (1 a n)
- MdN hace M tiradas de un dado de N caras
- *sum* C, suma todos los elementos de una colección C.
- *count* C, cuenta los elementos de una colección C.
- $+, -, *, /$ realiza operaciones aritméticas sobre los operandos.
- *sgn* y *mod*, función signo, módulo para poder representar mejor algunas operaciones.
- @ realiza una unión de los dos colecciones.

- $M < C$ retorna los elementos de C que son mayores a M. Válido para $>$, $>=$, $<=$, $==$, $!=$
- *min* C y *max* C encuentran el mínimo y máximo de una colección, respectivamente.
- *least* N C y *largest* N C, toma los N más chicos, o los N más grandes de una colección C

3.2.1 Ejemplos de operaciones básicas de tiradas aleatorias

- sum 2d10 + 3

  Suma dos tiradas de un dado de 10 caras y le agrega 3 al resultado.

- sum largest 3 4d6

  Toma las 3 tiradas más altas de 4 tiradas de un dado de 6 caras y las suma.

- count 7 < 6d10

  Cuenta cuantas de las 6 tiradas de un dado de 10 caras son mayores a 7.

- max 3d20

  Encuentra el máximo en 3 tiradas de un dado de 20 caras.

3.3 Operaciones complejas de tiradas aleatorias

Estas operaciones ya utilizan las operaciones básicas para darle todo el poder de representación que tiene el lenguaje para poder escribir de forma muy breve funciones que realizan las operaciones necesarias en diversos sistemas de tiradas.

- N # e hace N operaciones independientes de e y las combina en una colección.
- *if* C *then* $e1$ *else* $e2$ si C es una colección no-vacía, entonces realizá $e1$, si no, realizá $e2$.
- x := $e1$; $e2$ realizá $e1$ y lo guarda en un valor x, luego, para ocurrencia de x en $e2$, lo reemplaza
- *repeat* x := $e1$ *until* $e2$ repite las tiradas de $e1$ hasta que la expresión $e2$ evalue a algo no-vacío. Luego devuelve el último valor de $e1$.
- *accumulate* x := $e1$ *until* $e2$ repite las tiradas de $e1$ hasta que la expresión $e2$ evalue a algo no-vacío. Luego devuelve la unión de los valores de $e1$.

3.3.1 Ejemplos de operaciones complejas

- $1d4 \# 1d6$

  Evalúa una tirada de un dado de 4 caras, y realiza esa cantidad de tiradas de 6 caras.

- $b := 2d6 \; ; \; if \; (min \; b) = (max \; b) \; then \; b@b \; else \; b$

  Dado de Backgammon

- $repeat \; x := 2d6 \; until \; (min \; x) < (max \; x)$

  Repite tiradas de dos dados de 6 hasta que no saques dobles.

4 Cosas que puedo agregar

En el caso que esto no resulte suficiente como lenguaje, hay toda una rama diseñada dentro de Troll que puedo agregar que agregan versatilidad al lenguaje, al igual que otras operaciones de tiradas aleatorias. Alternativamente Troll admite la creación de funciones, el llamado, etc. Esto claramente agregaría un nivel de complejidad extra (junto a la necesidad de implementar un conjunto de operaciones, como componer funciones, entre otras).

4.1 Otras operaciones

Inicialmente existen otras operaciones que podría agregar a las ya descriptas, algunas de estas son relativamente "sencillas":

- $foreach \; x \; in \; e1 \; do \; e2$ evalúa $e1$, y por cada número n en el resultado, evalúa $e2$ con x reemplazando a n y luego une los resultados de e2.
- *drop* toma los elementos hallados en el primer argumento que no estén en el segundo.
- *keep* toma los elementos hallados en el primer argumento que estén en el segundo.
- *median* retoma la mediana de una colección
- $--$ retoma la diferencia entre dos multisets
- $1Zn$ o mZn funcionamiento similar a mdn pero con dados que comienzan en 0, en vez de en 1.

También hay un combinado de operadores lógicos como el and, or, implicancia, etc.

4.2 Funciones

En Troll se pueden crear funciones y funciones componibles, así como operadores como *call* para realizar sus llamados.

Estas funciones serían aceptables dentro de algunas de las operaciones anteriormente descriptas

4.3 Distribuciones de probabilidad

En Troll se representan las distribuciones de probabilidad utilizando conjuntos de pares (valor, probabilidad).

En Troll están impelmentadas las siguientes operaciones básicas:

- Un operador unario ! que le da una probabilidad de 1 de suceder a una
- Union de distribuciones de probabilidad
- Una operación ($d1$ $|p$ $d2$) que aplica sobre dos distribuciones, con una probabilidad p de elegir la distribución $d1$.

Además existe toda una sección que trabaja en la forma de representar gráficamente las distribuciones de probabilidad, así como la posibilidad de conseguir cierta información de esas distribuciones. Esta parte es un poco extensa en el paper, así que consideré innecesario expandirla si no iba a ser realizada.