Trabajo Práctico Final

Análisis de Lenguajes de Programación

Florencia Rovere

Introducción

Este trabajo se encarga de graficar coreografías pasadas al programa. La metodología de funcionamiento que tiene es la siguiente:

1. Se pasan como argumentos a la función *main* las opciones deseadas, descriptas más abajo, entre las cuales se aclara si se desean ingresar manualmente los pasos o utilizar un archivo de prueba.

2. Para cualquiera de las opciones elegidas, el programa realizará las funciones necesarias para graficar lo ingresado.

Organización de los archivos

Se entregan dos carpetas:

- **test/**: Se encuentran archivos de prueba. Como el nombre de cada uno lo indica, hay algunos de error y otros de éxito. Al compilar los archivos de error, se imprime en pantalla el debido mensaje, mientras que los otros grafican la coreografía.

- **src/**: Contiene el código que llevará a cabo la ejecución del trabajo. Los archivos que se encuentran son:

  - **Body.hs**: Es el encargado de graficar la coreografía. Consta de varias funciones, dentro de las cuales se encuentran:

    * *setup*: Inicializa el cuadro en el que se va a dibujar, especificando el tamaño, los marcos por segundo y la posición inicial del cuerpo.
    * *draw*: Dibuja un paso, que puede constar de un sólo movimiento o de varios. Llama a varias funciones auxiliares que grafican cada parte del cuerpo.
    * *update*: Actualiza los parámetros para dibujar el próximo paso. Como se explica en **Pasos de baile**, llamará a otras funciones auxiliares para tratar de manera distinta a los comandos **hacer n veces m**/**hacer n vez m** y **paso m**.

  - **DataType.hs**: Contiene los tipos de datos utilizados. Tenemos:

    * **Movement**: Determina la posición hacia la cual se moverá la parte del cuerpo indicada, ya sea *derecha, medio, izquierda, costado, arriba* o *abajo*.
    * **BodyPart**: Determina la parte del cuerpo elegida. Puede ser: *mano, pie, paso* o *cabeza*.
    * **States**: Determina un paso, la posibilidad de hacer un paso varias veces o el fin de la secuencia.
    * **Step**: Determina una línea de comandos (es decir, un paso). La misma puede constar de un único movimiento o de varios.
    * **ListSt**: Determina toda la coreografía ingresada.
    * **ActualPosition**: Determina un registro para llevar la posición actual de cada parte del cuerpo. Éste será actualizado cada vez que se lea un paso de baile y será leído cuando se desee graficar otro.

    * **Error**: Se utiliza para manejar errores.

- **Main.hs**: Arranca el trabajo. Funciones que contiene:
    * *main*: Llama a las funciones necesarias para comenzar el programa y analiza los argumentos que se le pasaron de entrada. Las opciones para éstos se detallan más adelante.
    * *drawing*: Llama a las funciones de **Body.hs** para graficar la coreografía.
    * *readHandler*: Verifica que no haya errores en el archivo ingresado.
- **Parser.hs**: Reconoce los pasos ingresados. Dentro de las funciones que se encuentran en este archivo, están:
    * *lets_dance*: Toma la secuencia de pasos ingresados por teclado.
    * *parserSt*: Divide cada línea de la secuencia en una lista de pasos si se ingresan más de un movimiento en el mismo paso. En caso de no ser así, se devolverá el único movimiento ingresado. Una vez que se obtiene el resultado de hacer esto, se parseará cada elemento de la lista con la función *recognizeList*, la cual a su vez llamará a las debidas funciones auxiliares.
    * *recognize_mistakes_list*: Determina si hay errores en la coreografía para devolver el mensaje correspondiente.
    * *eval_list*: Quita de cada elemento el constructor *Just*. No toma en cuenta el caso que sea *Nothing* dado que esto ya fue chequeado en la función *recognize_mistakes_list*.
- **ParserMain.hs**: Reconoce los argumentos de la función *main*. Tiene diferentes estructuras:
    * **Parameters**: Tipo de datos para las opciones disponibles para *main*.
    * **Err**: Se utiliza para manejar errores.
    * **R_Param**: Registro para llevar las opciones ingresadas por el usuario.
    * *getting_right_parameters*: Comienza el parseo de los argumentos ingresados y genera el registro con las opciones ingresadas. En caso de que alguna no esté especificada, se tomarán los valores por defecto.
    * *recognize_parameters*: Parsea cada argumento del usuario.
    * *recognize_mistakes*: Determina si hay errores en los argumentos para devolver el debido mensaje.
    * *eval*: Quita de cada elemento el constructor *Just*. No toma en cuenta el caso que sea *Nothing* dado que esto ya fue chequeado en la función *recognize_parameters*. Actualiza el registro con los valores ingresados.
    * *float*: Parsea un número flotante. Ver la sección **Código ajeno**.
- **States.hs**: Define una mónada de estados y una función *update* que actualiza los valores de la misma.

Argumentos de la función Main

Main puede tomar 4 parámetros:

- **opt**: Junto con un número entero (1 o 2) decide la opción que se ingresará:
  - *opt1*: Los pasos serán ingresados manualmente.
  - *opt2*: Los pasos se tomarán de un archivo de prueba.
- **fr**: Junto con un número flotante determina los marcos por segundo. Esto especificará la velocidad con que se grafiquen los pasos. Si no se ingresa esta opción, se utilizará por defecto *fr1*.
- **n1xn2**: Siendo **n1** (longitud) y **n2** (altura) enteros mayores a 200, establece el tamaño de la ventana en que se graficará. Si no se ingresa esta opción, se utilizará por defecto *400x400*.
- **filestr**: Siendo **str** un archivo o ruta de archivo, lo utilizará como entrada si se ingresó la opción *opt2*. Un ejemplo sería *file../test/test1.txt*. Es importante aclarar que entre el identificador y los argumentos que decide entrar el usuario, no debe haber espacios dado que esto no permitirá que los mismos se reconozcan correctamente. Por identificador se entiende a las palabras **opt**, **fr**, **x** y **file**, y por argumentos se entiende a los números enteros y flotantes y al archivo. Un ejemplo de main que ingrese todas las opciones será:

  > ./Main opt2 fr2.5 file../test/test6.txt 600x400

  El cual tomará el archivo de entrada ../test/test6.txt y lo ejecutará con 2.5 marcos por segundo en un cuadro de 600 x 400.

Pasos de baile

Tanto al ingresar los pasos manualmente como al utilizar archivos de prueba, las opciones que se pueden ingresar son:

- **mano m s**: Esta opción moverá el brazo indicado con **m**, que puede tomar los valores *derecha* o *izquierda*, hacia la posición indicada con **s**, que puede tomar los valores *abajo*, *costado* o *arriba*.
- **pie m s**: Análogamente a la mano, esta opción moverá la pierna indicada con **m**, que puede tomar los valores *derecho* o *izquierdo*, hacia la posición indicada con **s**, que puede tomar los valores *abajo* o *costado*.
- **cabeza m**: Esta opción moverá la cabeza hacia donde se le indique con **m**. Este parámetro tomará los valores *derecha*, *medio* o *izquierda*.
- **paso m**: Esta opción realizará un paso hacia el lado indicado con **m**, ya sea *derecho* o *izquierdo*. Lo realiza en dos tiempos: en el primero mueve únicamente la pierna hacia el lado donde se trasladará y en el segundo efectivamente realiza el movimiento. Es por esto que al actualizar los valores en el archivo **Body.hs** se considera como un caso particular.
- **hacer n veces m** o **hacer n vez m**: Esta opción repetirá el paso **m** tantas veces como lo indique el entero **n**. **m** puede ser cualquiera de los pasos descriptos anteriormente. Este comando se comporta de una manera particular si es ingresado con otros dentro de la misma línea:

  - Si es el primero de la línea, se ejecutará completamente y en el último movimiento que le corresponde, también actualizará los otros movimientos ingresados en el mismo paso.
  - Si hay algún movimiento antes que éste, tomará el comando como si fuera **hacer 1 vez m** y ejecutará un sólo movimiento de **m** junto con los otros ingresados en la misma línea. Esto se debe a que intenta actualizar todos los valores en paralelo, y desecha el resto de los movimientos para poder finalizar con dicho paso en una actualización.

- **fin**: Cuando se desee llegar al fin de la coreografía, se debe ingresar esta opción para indicar que la misma ha terminado.

En caso de ser un único movimiento, se ingresa el mismo y se presiona la tecla *enter*. En caso de querer ingresar varios movimientos juntos, los mismos deben estar separados por coma (,) y para pasar al próximo paso, nuevamente se presiona la tecla *enter*.
Un pequeño ejemplo es:

mano derecha arriba, pie izquierdo costado
paso derecho
hacer 4 veces cabeza derecha
fin

Este ejemplo moverá primero la mano derecha hacia arriba y el pie izquierdo hacia el costado simultáneamente. Luego realizará un paso a la derecha y por último movera la cabeza hacia la derecha cuatro veces.

Requisitos

Se deben tener instalados los paquetes **Processing** y **Parsec**.

- **Processing**: Se puede instalar con
  > cabal install processing-for-haskell

- **Parsec**: Se puede instalar con
  > cabal install parsec

Cómo compilar

Una vez que estén instalados los paquetes, sencillamente se compila con ghc. Es decir que, estando en la carpeta **src/** se deben ingresar los siguientes comandos:

> ghc Main.hs
> ./Main **opciones explicadas en **Argumentos de la función main****

Processing

Processing es un software para aprender a codificar dentro del contexto de las artes visuales.
Processing for Haskell es un EDSL para realizar gráficos. Es una librería de animación que implementa un subconjunto del lenguaje Processing en Haskell. No está terminado aún sino que se sigue trabajando para representar las funciones que faltan. [2]

Código ajeno

Se utilizó una función que parsea flotantes ya definida. La original se puede encontrar en [8].
Además, ideas sobre cómo organizar el código para graficar los pasos de baile fueron tomadas de [2].

Bibliografía

[1] https://processing.org
[2] https://github.com/anton-k/processing-for-haskell
[3] https://wiki.haskell.org
[4] https://hackage.haskell.org
[5] https://hackage.haskell.org/package/parsec
[6] https://www.haskell.org/
[7] https://www.haskell.org/hoogle
[8] https://www.schoolofhaskell.com/school/to-infinity-and-beyond/pick-of-the-week/parsing-floats-with-parsec