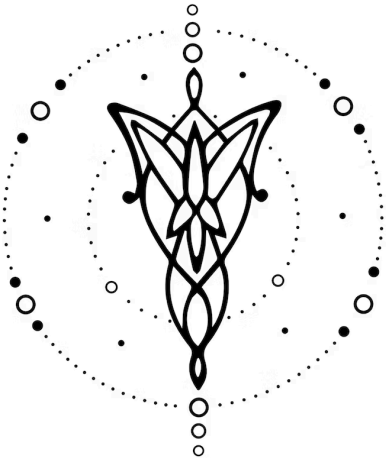

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

1. Equipo 1
2. Repositorio 1
3. Dominio 2
4. Construcciones 2
5. Casos de Prueba 3
6. Ejemplos 3

1. Equipo

Nombre	Apellido	Legajo	E-mail
Lucas	Campoli	64.295	lcampoli@itba.edu.ar
Franco Agustín	Ghigliani	63.312	fghigliani@itba.edu.ar

2. Repositorio

La solución y su documentación serán versionadas en: TLA_TP-MLang.

3. Dominio

Desarrollar un lenguaje orientado a objetos que permita modelar y crear sistemas dinámicos de videojuegos como game-loops, sistemas de economía, etc. Estos sistemas luego pueden ser importados desde machinations.io para ser simulados. Desde este sitio, también pueden ser exportados a Unity para el desarrollo de videojuegos. El lenguaje debe permitir establecer las propiedades de los diferentes nodos y sus conexiones, y definir los parámetros generales de la simulación como su velocidad, la cantidad de steps a simular, entre otros.

Para reducir la complejidad del proyecto, no todos los posibles componentes de Machinations serán representados en el lenguaje, tales como traders o registros. Se tomó esta decisión ya que son componentes cuyos usos no son comunes ni frecuentes, y además no son necesarios para el desarrollo de una simulación suficientemente compleja. Tampoco se tuvo en cuenta los aspectos relacionados al estilo, como pueden ser los colores o aspectos de los componentes en la interfaz gráfica de Machinations.

La implementación satisfactoria de este lenguaje permitiría experimentar con diferentes sistemas comunes en videojuegos y sus variaciones de implementación, medirlos y balancearlos antes de implementar los mismos en un proyecto, minimizando tiempos de playtesting y facilitando la detección de sistemas desbalanceados.

4. Construcciones

El lenguaje desarrollado debería ofrecer las siguientes construcciones, prestaciones y funcionalidades:

(I). Se podrán crear una o varias redes, y cada una tendrá nombre, *default gateway* y rango de direcciones en formato IPv4 (*e.g.*, una red en el rango `10.1.0.0/16`).

(II). Se podrán crear uno o varios *hosts* dentro de cada red, y cada uno tendrá un nombre y dirección IP asignada.

(III). Se podrán agregar puertos de entrada y/o salida dentro de cada *host*.

(IV). Se podrán crear paquetes para enviar a través de los puertos de un *host*, donde cada paquete contiene una o varias variables.

(V). Las variables podrán ser de tipo *packet*, *address*, *integer*, *boolean* o *string*.

(VI). Las variables podrán ser vectores de alguno de los tipos anteriores.

(VII). Se proveerán operadores relacionales como <, >, =, ≠, ≤ y ≥.

(VIII). Se proveerán operaciones aritméticas básicas como +, -, * y /.

(IX). Se proveerán operaciones lógicas básicas como AND, OR y NOT.

(X). Se proveerán estructuras de control básicas de tipo IF-THEN-ELSE, FOR y WHILE.

(XI). Se podrá definir un pequeño procedimiento sobre cada puerto dentro de un *host* que manipule los paquetes recibidos y/o emita otros en respuesta.

(XII). Se podrán emitir paquetes desde un host hacia otro.

(XIII). Se podrán crear tablas de ruteo dentro de cada *host*.

5. Casos de Prueba

Se proponen los siguientes casos iniciales de prueba de **aceptación**:

(I). Un programa que comunique 3 redes a través de un *router*.
(II). Un programa que construya 2 redes, una pública y otra privada.
(III). Un programa que construya un 1 *host* dentro de una red.
(IV). Un programa que construya y asigne 1 (una) tabla de ruteo.
(V). Un programa que exponga un servicio en un *host*.
(VI). Un programa que exponga un servicio en un *host*.
(VII). Un programa que filtre paquetes en un *host*.
(VIII). Un programa que implemente un balanceador de carga entre 2 *hosts*.
(IX). Un programa que implemente un *proxy* entre 2 redes.
(X). Un programa que manipule el contenido de los paquetes.

Además, los siguientes casos de prueba de **rechazo**:

(I). Un programa malformado.
(II). Un programa que asigne 2 servicios diferentes, en el mismo *host* y puerto.
(III). Un programa que asigne un *host* a una red que no existe.
(IV). Un programa que construya 2 redes con espacios de direcciones solapadas.
(V). Un programa que intente operar entre tipos de datos incompatibles.

6. Ejemplos

Creación de una red, con un *host* que ofrece una funcionalidad similar al protocolo ECHO en el puerto 7 (*i.e.*, devuelve el mismo paquete que recibe):

```
// Crear una nueva red que soporte hasta 254 hosts:
network UnaRedDePrueba over 10.0.0.0/8;

// Crear un host dentro de la red anterior:
host UnHostConEcho in UnaRedDePrueba;

// Crear un servicio ECHO dentro del host:
add listening port 7 to UnHostConEcho {
    return packet;
}
```

Crear un *firewall* que bloquee todos los paquetes que provienen desde el *host* con dirección IPv4 `192.168.1.45`, pero permitiendo la propagación del resto:

```
// Crear un firewall dentro de la red del ejemplo anterior:
host UnFirewall in UnaRedDePrueba;

// Permitir que todos los paquetes pasen, excepto desde cierta IP:
add forwarding to UnFirewall {
    if (packet.source.ip = 192.168.1.45) {
        log “Se bloqueó el siguiente paquete desde {2}: {1}”, packet, host.ip;
        drop;
    }
    else {
        return packet;
    }
}
```