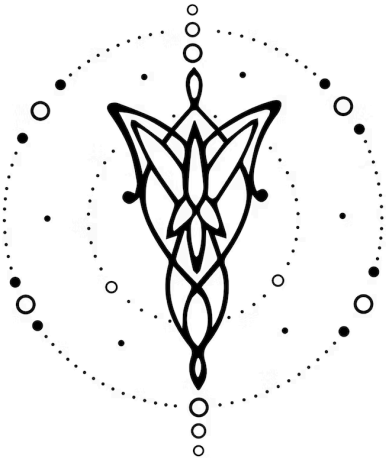

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

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 declarativo que permita modelar y crear sistemas dinámicos de videojuegos como game-loops, sistemas de economía, sistemas de niveles del jugador, 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 las integraciones relacionadas 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, tales como diseñar economías o procesos aleatorios que ocurren dentro de 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 uno o varios sistemas para ser simulados en simultáneo.

(II). Se podrán crear nodos de los tipos *Source*, *Converter*, *Gate*, *Drain*, *Pool*, *Delay*, y *EndCondition*. Estos tendrán los atributos básicos:

- (II.1). *label*: el nombre a mostrar en la simulación
- (II.2). *position*: coordenadas (x, y) en las cuales mostrar el nodo al ser importado desde Machinations
- (II.3). *activation*: cuándo activar el nodo. Podrá ser *interactive* (activa con click), *passive* (activa con señales de otro nodo), *automatic* (activa en cada step), o *onstart* (activa únicamente en el primer step). La única excepción es el nodo *EndCondition*, el cual no tendrá este atributo.

(III). Los *Sources* serán los generadores de recursos. Además tendrán un atributo adicional *resourceColor*.

(IV). Los *Converters* transformarán recursos -definidos por las conexiones entrantes- en otros. También tendrán los atributos:

- (IV.1). *resourceColor*
- (IV.2). *conversion*

(V). Los *Gates* distribuirán los recursos entrantes entre los nodos conectados. La distribución de los recursos será determinada por las conexiones de recursos salientes. Además tendrán los atributos

- (V.1). *activationMode*
- (V.2). distribution

(VI). Los *Drains* se desharán de recursos cual tacho de basura. También tendrán el atributo *activationMode*

(VII). Los *Pools* serán contenedores de recursos y servirán como medición de diversas cosas como, por ejemplo, cuántas monedas tiene el jugador en la simulación. Tendrán además atributos:

- (VII.1). *activationMode*
- (VII.2). *initialResources*
- (VII.3). *initialResourcesColor*
- (VII.4). *capacity*
- (VII.5). *numberDisplayThreshold*
- (VII.6). *overflow*

(VIII). Los *Delays* servirán para retrasar el tránsito de recursos. Tendrán también el atributo adicional *queue*

(IX). Finalmente, los *EndConditions* al ser activados detendrán la simulación. Representarán un fin del juego (sea porque el jugador pierde, porque gana, o porque simplemente no tiene sentido continuar la simulación). No tendrán ningún atributo adicional pero la condición que representan estará definida por la fórmula de la conexión de estado entrante.

(X). Mediante métodos predefinidos, se podrán crear las conexiones entre los nodos para simular transferencias de recursos o condiciones.

(X.1). Mediante conexiones de recursos, se simularán las transferencias de recursos entre los distintos nodos.

(X.2). Mediante conexiones de estado, se podrán mandar las señales para activar un nodo con activation=passive o alterar la fórmula de una conexión de recurso.

(XI). Se podrá crear *Templates* nodos a partir de un tipo de nodo preexistente. Es decir, definir los atributos de un nodo de ese tipo para luego poder ser reutilizado. También servirá para centralizar estos parámetros ya que al modificar los parámetros especificados en el *Template* se modificarán todos los nodos creados a partir del mismo.

(XII). Se podrá sobrescribir parámetros de un template en la declaración de un nodo de ese tipo.

(XIII). Se podrá marcar el resto de una línea como un comentario con la keyword //

(XIV). Se podrán definir constantes con la keyword *const* para luego ser utilizada en los parámetros de los distintos objetos

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;
    }
}
```