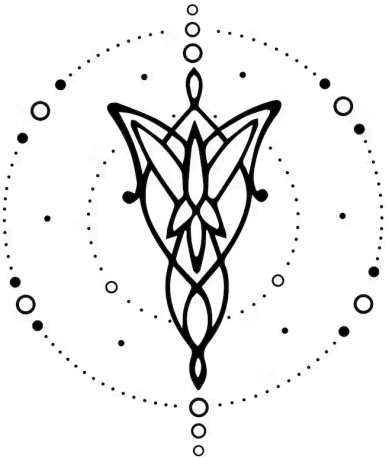

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

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

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, cada una con sus argumentos:
- (I.1). *name*
- (I.2). *stepInterval*
- (I.3). *stepsToSimulate*

(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. (este campo será opcional pues de no usarlo, la posición será calculada automáticamente)
- (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*, lo que permite diferenciar los recursos para representar distintas cosas.

(IV). Los *Converters* transformarán recursos -definidos por las conexiones entrantes- en otros. También tendrán los atributos:
- (IV.1). *resourceColor*
- (IV.2). *multipleConversion*

(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). *randomDistribution*

(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). *drainOnOverflow*

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

(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 el uso de "=>" o "=#", se podrán crear las conexiones entre los nodos para simular transferencias de recursos o condiciones respectivamente.

(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* de 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). A su vez, se podrá crear *Templates* de *Simulations* (circuitos enteros) especificando entradas y salidas. Estos *Templates* de *Simulations* podrán ser reutilizados como si fuera un nodo para tener su mismo comportamiento repetido en diversas partes de una misma simulación.

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

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

(XV). 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 cree una simulación de entrenamiento de tropas a partir de un pago de una cantidad de recursos definida.
(II). Un programa que cree una simulación del ciclo de recursos. Desde su obtención, hasta la creación de un ítem.
(III). Un programa que cree una simulación de la ganancia de experiencia del jugador.
(IV). Un programa que cree una simulación de un game-loop.
(V). Un programa que cree una simulación del sistema de economía del juego.
(VI). Un programa que cree una simulación del sistema de regeneración de vida del jugador.
(VII). Un programa que cree una simulación de refinamiento de distintos recursos a otro de otro tipo.
(VIII). Un programa que cree una simulación de un control de tránsito con semáforos.
(IX). Un programa que cree una simulación para un contador de combos.
(X). Un programa que cree una deconstrucción de un sistema preexistente de otro juego (por ejemplo: el sistema de cansancio y hambre de Minecraft).

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

(I). Un programa malformado.
(II). Un programa que cree una conexión de recurso desde un *Pool* **hacia** un *Source*.
(III). Un programa que asigne una cadena de texto como fórmula para una conexión.
(IV). Un programa que intente asignar un valor incorrecto a una propiedad de un nodo.
(V). Un programa que intente modificar constantes.
(VI). Un programa con una *Simulation* que se utilice como *Template* en sí misma, generando una recursión.

6. Ejemplos

Crear una simulación de un sistema de regeneración de vida del jugador (la simulación se puede probar aquí).

```
Simulation healthRegenSim (name="Health Regeneration", stepInterval=1,
                           stepsToSimulate=100){
    Source regen {
        label = "Regeneration";
        position = (-690, 205);
        activation = automatic;
        resourceColor=Red;
    }

    Source hp_potion {
        label = "HP Potion";
        position = (-287, 207);
        activation = interactive;
        resourceColor = Black;
    }

    Drain dmg {
        label = "Damage";
        position = (-470, 365);
        activation = passive;
        activationMode = pullAny;
    }

    Pool hp {
        label = "HP";
        position = (-470, 205);
        activation = passive;
        activationMode = pushAny;
        initialResources = 10;
        initialResourcesColor = Red;
        capacity = 20;
        numberDisplayThreshold = -1;
        drainOnOverflow = false;
    }

    Gate dmg_trigger {
        label = "";
        position = (-621, 368);
        activation = automatic;
        activationMode = pullAny;
        randomDistribution = true;
    }
```

```
    regen => hp {1};
    hp => dmg {3};
    hp_potion => hp {5};

    hp =# regen {<5};
    dmg_trigger =# dmg {35%};
}
```

Crear una simulación de minado (acceda a la simulación aquí), en la cual cada vez que se pica un curso hay una probabilidad de que te dropee carbón o hierro (no necesariamente las probabilidades tienen que sumar 100%), y luego los recursos se utilizan para forjar espadas:

```
const eight = 8;
Simulation sim (name="Mining and Crafting", stepInterval=1, stepsToSimulate=100){

      Template:Pool Almacenamiento {
            activation=passive;
            activationMode=pull-any;
            initialResources=0;
            initialResourcesColor=Black;
            capacity=-1;
            numberDisplayThreshold=0;
            drainOnOverflow = false;
      }

      Source mine {
            label="Mina de recursos";
            activation=automatic;
            resourceColor=Black;
            position=(0, 0);
      }

      Gate type{
            label="Tipo de recurso";
            activation=passive;
            activationMode=pull-any;
            randomDistribution=false;
            position=(50, 0);
      }


      new Almacenamiento coal {
            label="Carbón almacenado";
            position=(100, 50);
      }
```

```
        new Almacenamiento iron {
                label="Hierro almacenado";
                position=(100, -50);
        }

        Converter forge {
                label="Forja";
                activation=automatic;
                resourceColor=Black;
                multipleConversion=false;
                position=(150, 0);
        }

        new Almacenamiento swords {
                label="Espadas creadas";
                position=(200, 0);
        }

        //Conecto la mina con la gate, y la gate distribuye a los almacenamientos
        en base a la probabilidad
        mine => type {1};
        type => coal {60%};
        type => iron {20%};

        //Conecto los almacenamientos con la forja, y solicitó que para forjar
        una espada se necesitan 3 carbones y 1 hierro
        coal => forge {3};
        iron => forge {1};

        //Conecto la forja al almacenamiento de espadas (se podría agregar un
        delay en el medio para simular tiempo de crafteo)
        iron => forge {(1 + 2) * 3 - eight};
}
```