Propuesta a Lenguaje de Road to Civilization

La idea central es que sea un lenguaje parecido a C#, donde puedas hacer en primera instancia las cosas básicas de este lenguaje.

Los tipos del lenguaje son: *Number, String, Boolean, List, Species, Society, Land*. Además se tiene un tipo padre Simulation invisible al usuario en donde se tienen todas las funciones build_in. Los tipos empiezan siempre con una letra mayúsculas

```
Type name = value;
```

Será la sintaxis para declarar una variable, las que siempre se deben empezar con una letra minúscula

Las entidades(Species, Land, Society) se declaran:

```
Type name = Type(value_1, value_2, …);
```

Que básicamente indica que lo que se guarda en name es una intancia de la entidad Type.

La variables se pueden asignar luego sin especificar el valor de retorno

```
name = value;
```

Las listas se pueden asignar o indexar de la siguiente forma

```
List a = [1,3];
Number b = a[0];
```

No se tendrán comentarios.

Las estructuras de control serán iguales:

```
if(conditions)
{       Code;    }
else
{       Code;    }
```

Las condiciones pueden usar indistintamente "and", "or" y "not", y los comparadores serían <, > y ==

Entonces, tendremos igualmente el while y for trabajando de la misma forma:

```
while(conditions) {         Code;       }
```

Las funciones se declaran especificando el tipo de las variables de retorno, así como el tipo de las variables de entrada se devuelve la última línea la que debe ser una asignación de una variable que tenga el mismo tipo de retorno, los nombres de las funciones deben empezar simpre con _

```
Type _name(Type_1 value_1, Type_1 value_2, …)
{
        Code;
        Type return = value;
        Code;
        return = return_value;
}
```

Ahora, con respecto a la simulación en general.
Para crear una nueva especie se necesita lo siguiente:

```
Specie name = Species(“Especie_name”);
```

Recibe por argumento el string del nombre

Para hacer un nuevo Land se necesita lo siguiente:

```
Land  name = Land([value,value])
```

Recibe por argumento la posición en la que las quieres poner

Y para Sociedad:

```
Name = Society(“Society_name”, specie)
```

Recibe por argumento el string del nombre y la instancia de la especie a la que pertenece la sociedad.

Estos tipos tienen funciones para trabajar sus caracteristicas, dependencias e influencias, para ello se tienen las funciones:

- `_changeCharacteristic`: Añade o cambia una caracteristica, recibe por entrada nombre de la misma, valor, límites(inferior y superior), mutabilidad y función de distribución.
- `_deleteCharacteristic`: Elimina una caracteristica, recibe por entrada nombre de la misma
- `_getCharacteristic`: Toma el valor de una caracteristica de una entidad, recibe por entrada nombre de la misma
- `_getCharacteristicSummation`: Toma el valor de la sumatoria de un caracteristica de las sociedades que pertenecen a una especie, recibe por entrada nombre de la misma
- `_getCharacteristicMean`: Toma el valor de la media de un caracteristica de las sociedades que pertenecen a una especie, recibe por entrada nombre de la misma
- `_addDependence`: Añade una dependencia(solo los land), recibe por entrada el nombre de la entidad y caraccteristica de a, el nombre de la entidad y caracteristica b, c y las funciones plus y mult
- `_addInfluence`: Añade una influencia(solo los land), recibe por entrada el nombre de la entidad y caraccteristica de a, el nombre de la entidad y caracteristica b, c y las funciones plus y mult
- `_deleteDependence`: Elimina una dependencia(solo los land), recibe por entrada el nombre de la entidad y caraccteristica de a, el nombre de la entidad y caracteristica b
- `_deleteInfluence`: Elimina una influencia(solo los land), recibe por entrada el nombre de la entidad y caraccteristica de a, el nombre de la entidad y caracteristica b

Su sintaxis es:

```
land._addDependence("entity_1","charact_1","entity_2","charact_2",c, _plus, _mult);
```

Una vez dicho esto pasamos directamente a cómo se define la simulación.
Siempre va a existir una simulación llamada directamente Simulation, que es la simulación principal.

Las comandos para interactuar con Simulation son:
`_redimension(value_1, value_2);`, donde ambos parámetros deben ser int.
`_start();`, no recibe parámetros e inicia la simulación
`_end(cond)`, esta será explicada más adelante

Igualmente está una biblioteca que implementamos por detrás, la biblioteca `_random("normal")`, recibe por entrada el string con el nombre de la distribución a ejecutar, estas reciben dos parámetros por entrada estas funciones, que son el límite inferior y superior, o media y variancia, y devuelven el valor del random descrito evaluado en ese intervalo.
Ejemplo:

```
Number a = _random("normal",[3,5])
```

Las distribuciones que se pueden pedir son: "uniform", "exponential", "gamma", "normal", "binomial", "geometric".

Quedando esto dicho, vamos ahora a ver qué podemos hacer mientras corre una simulación.

Es posible declarar código que trabajará mientras corre la simulación, esto se hacer con la función `_day` de la siguiente manera:

```
_day(cond, _function)
_day(_cond, _function)
```

Donde esto lo que hace es que mientras se ejecute la simulación por cada día que se cumpla la condición se ejecuta la función.

Entonces, no se puede escribir comando en estas partes, solo los comandos del tipo:
`_write(value)`, que recibe un valor que puede ser bien dentro de la simulación, por ejemplo:

```
_write(human._getCharacteristic("Pobalción"));
```

Y lo imprime en consola
`_record(name,value)`, crea un archivo txt de nombre name y lo modifica guardando dentro el nombre de la variable value y su valor (en caso de existir el txt solo se agrega la información)
`_end(condición)`, la cual ya habíamos mencionado anteriormente, recibe una función o una condición:

```
_end(human._getCharacteristic("Population")>100 and cows._getCharaccteristic("Population")>50)
_end(_cond_function)
```

Donde la Simulación termina en este momento.

Para añadir/eliminar entidades y dependencias a la simulación se tienen las funciones:

- `_addSociety`
- `_addSpecies`
- `_addLand`
- `_deleteSociety`
- `_deleteSpecies`
- `_deleteLand`
- `_addInterdependences`
- `_deleteInterdependences`
- `_addInfluences`
- `_deleteInfluences`

Para modificar si una sociedad tiene la capacidad de evolucionar se usa la función `enableEvolution`.

y para las funciones de distribución y suma y multiplicación se tienen las funciones ya implementadas.

- `_distribution`
- `_plus`
- `_multiplication`

Aunque el usuario pude implementar sus propias funciones para usarlas en su lugar.

Para obtener la cantidad de días que han pasado se usa la función `actualDay` que no recibe entrada.

Se puede obtener un string de objetos de tipo *Number, Boolean, List* usando las funciones:

- *_numberToString*
- *_booleanToString*
- *_listToString*

se puede obtener el tamaño de una lista con el método: `getLenght`