Documentación POUEX

- *Santiago Rangel Colón*
- *Samuel Moreno Vincent*

Contenido

Consideraciones previas a la ejecución:

Nuestro proyecto tiene dos modificaciones opcionales introducidas además de varias funcionalidades especiales que es importante conocer antes de ejecutarlo, las dos modificaciones opcionales son incluir una interfaz gráfica para el proyecto y que el juego permita guardar y cargar partida, debajo explicamos cada una de las modificaciones y funcionalidades que hemos incluido en nuestro proyecto:

Nuestra interfaz tiene un gif con una mascota, para poder ver este gif es necesario tener el archivo Poux.gif y Poux_death.jpg dentro de la carpeta pictures de la carpeta del proyecto java o en una carpeta llamada pictures en el mismo directorio que el archivo .jar en el caso de haberlo generado. Junto al proyecto adjuntaremos una carpeta llamada Pouex.java en la que estará la carpeta pictures necesaria, y además adjuntaremos la carpeta con las imágenes aparte por si se quieren adjuntar a otro proyecto.

Al pulsar el botón guardar y cargar de la interfaz generará o cargara un archivo llamado pouex.save en el momento de pulsarlo, este archivo se generará o cogerá de la una carpeta llamada saves del directorio del proyecto java o un directorio llamado igual si está en el mismo directorio que el archivo .jar. Para poder generar el guardado es necesario que la carpeta saves exista.

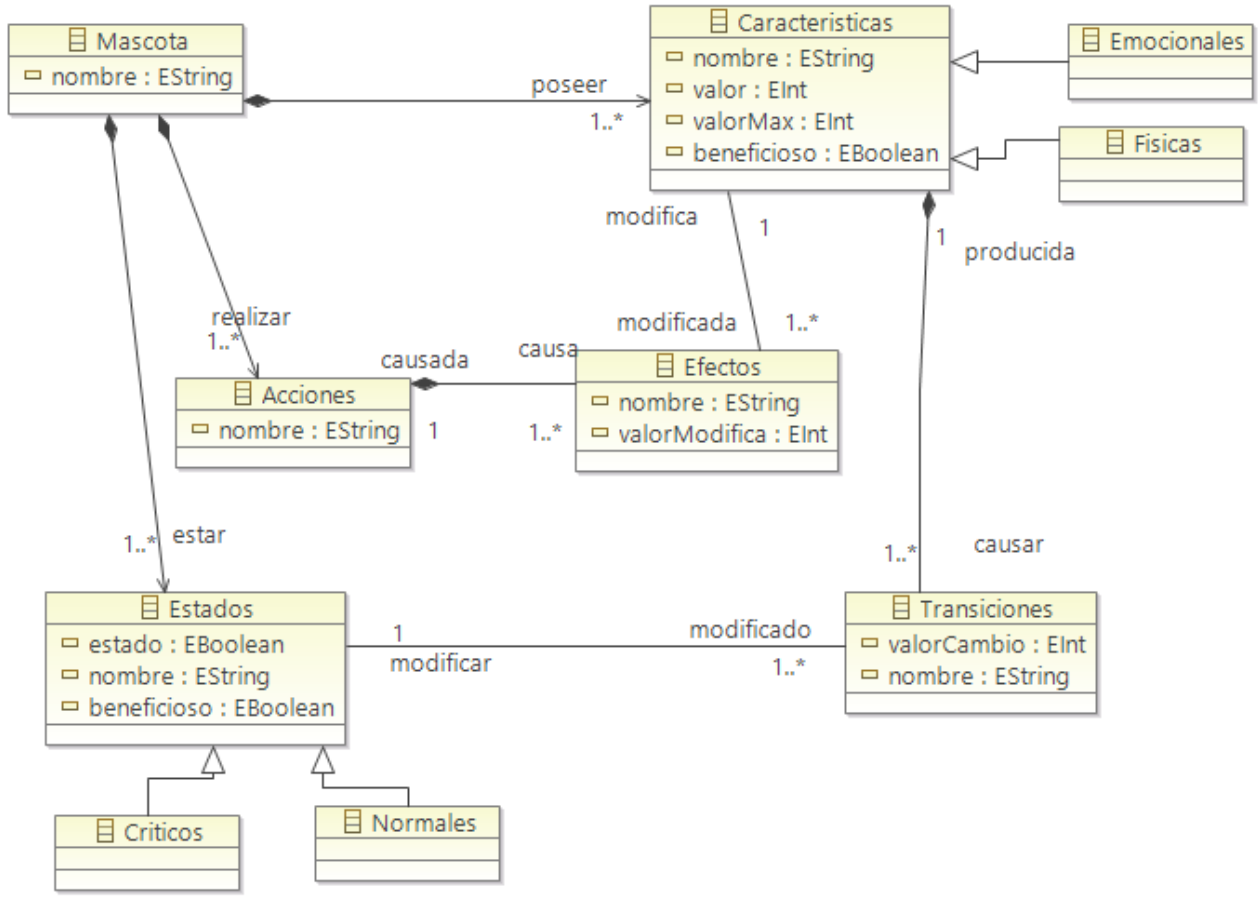
Tiempo: Si creamos la acción tiempo esta no actuara como una acción normal, en el caso de que el programa generado tenga una acción llamada tiempo esta se ejecutara automáticamente cada 15 segundos desde el comienzo de la aplicación, en el caso de que esta acción tiempo no se cree en el modelo, no se ejecutará la acción tiempo automáticamente, solo funciona el tiempo si es creado. Para que la acción tiempo funcione correctamente deberá llamarse en el modelo tiempo o Tiempo.

Nuestros estados pueden ser activados o desactivados por distintas transiciones, por ejemplo es posible tener un estado llamado Muerte que sea activado cuando una característica alcance un cierto valor, este estado muerte podrá ser activado cuando las características que queramos superen el valor de su transición a muerte, de esta forma no es necesario tener estados como MuertexSucieda y MuertexTristeza, es posible tener un estado Muerte que se active cuando

cualquiera de las dos características alcance el valor que lo hace morir, también es posible tener estas muertes por separado, queda a elección de cada uno como quiere que sea en su aplicación.

Debido a restricciones para el correcto funcionamiento de nuestra aplicación no se puede poner un valor inicialmente superior a su valorMax, al igual que no podemos tener valor Cambio de las transiciones con un valor superior o igual al valorMax de la característica con la que está relacionada la transición, esto esta hecho porque activamos algunos estados al ser superados por lo que si podemos tener valores de cambio iguales al valorMax no podrá aumentar más y por lo tanto no cambiaría de estado nunca, por esta razón si quieres que un estado se active al llegar la característica a 10, el valor Cambio deberá ser de 9 siempre que la característica con la que está relacionada tenga un valorMax de al menos 10.

Metamodelo del dominio:



Conceptos del dominio

- **Mascota**
    - Descripción: Es nuestra clase principal, todo gira en torno a la mascota, la mascota realizará acciones, poseerá características y se encontrará en estados.
    - Atributos:
        1. nombre: String que almacenará el nombre de una determinada acción.
    - Referencias a otros conceptos y su justificación:
        1. Poseer: Es la referencia entre la mascota y sus características, nuestra mascota puede poseer de 1 a n características, estas características son los atributos de nuestra mascota y variarán a lo largo de la ejecución de la aplicación. Es necesaria para poder almacenar el conjunto de características de la mascota.
        2. Realizar: Es la referencia entre mascota y sus acciones, nuestra mascota puede realizar de 1 a n acciones, estas acciones son las cosas que puede realizar nuestra mascota y causara efectos que variarán las características de la mascota. Es necesaria para poder acceder a las acciones que podrá realizar nuestra mascota.
        3. Estar: Es la referencia entre mascota y sus estados, nuestra mascota puede estar de 1 a n estados, los estados están causados cuando algunas características alcanzan determinados valores activando el estado de la mascota mientras la característica no deje de estar en el rango de valores que activan el estado. Es necesaria para poder acceder a los diferentes estados en los que se puede encontrar la mascota.
    - Restricciones:
        1. El atributo nombre no debe estar vacío.
- **Acciones**
    - Descripción: Las acciones son las actividades que puede realizar nuestra mascota, estas acciones causarán efectos que modificaran el valor de las características por lo que las acciones son las causantes de las variaciones en las características de la mascota.
    - Atributos:
        1. nombre: String que almacenará el nombre de la acción.
    - Referencias a otros conceptos y su justificación:
        1. Causa: Es la referencia entre la acción y sus efectos, la acción puede causar de 1 a n efectos, estos efectos son los causantes de los cambios en las características de nuestra mascota. Es necesario que las acciones creen efectos para poder decidir en qué cantidad modificaran las características.

- Restricciones:
    1. El atributo nombre no debe estar vacío.
    2. Una acción no puede generar 2 efectos que afecten a una misma característica.

- **Características:**
    - Descripción: Son las diferentes cualidades que posee nuestra mascota, estas cualidades son modificadas por los efectos de las acciones y son las causantes de las transiciones entre un estado y otro. De la clase Características heredan dos subclases, Emocionales y Físicas, las características físicas son aquellas que reflejan un estado del cuerpo de la mascota mientras que las emocionales dependen del estado mental de la mascota.
    - Atributos:
        1. nombre: String que almacenará el nombre de la característica.
        2. valor: Entero que guarda el valor actual de una determinada característica.
        3. valorMax: Entero que guarda cual es el valor máximo que puede alcanzar una característica.
        4. beneficioso: Booleano que guarda información de si una característica es buena o mala, es decir, si una característica beneficiosa es superior al valor de cambio activara un estado beneficioso (Feliz) o desactivará un estado no beneficioso (Triste), al contrario sí una característica es no beneficiosa al superar el valor de cambio de una determinada transición activara un estado no beneficioso (Triste) o desactivará un estado beneficioso.
    - Referencias a otros conceptos y su justificación:
        1. Modificada: Es la referencia entre la característica y los efectos que la modifican, la característica puede ser modificada por 1 o n efectos, estas efectos son causados por las acciones de la mascota y modifican el valor de la característica. La función de esta relación es para poder controlar que una característica puede ser modificada por más de un efecto.
        2. Causar: Es la referencia entre la característica y las transiciones que podría causar la característica, la característica puede causar de 1 a n transiciones, las transiciones son las causantes de activar o desactivar un estado y dependen del valor que tenga una característica. La función de esta relación es necesaria para crear las diferentes transiciones que dependen de una característica.
    - Restricciones:
        1. El atributo nombre no debe estar vacío.
        2. El campo valor no debe ser mayor o igual que 0 y debe ser menor que valorMax y no estar vacío.
        3. El campo valorMax debe ser mayor que 0 y no estar vacío.
        4. El valor no puede ser mayor que el valorMax.

- **Estados**
    - Descripción: Los estados son situación actual de la mascota, estos estados son activados por las transiciones que dependen del valor de una determinada característica. Hay dos clases que heredan de la clase Estados, Críticos y Normales, los estados críticos son especiales ya que pueden causar la muerte que produciría la finalización del juego o un estado con un efecto similar.
    - Atributos:
        1. nombre: String que almacenará el nombre del estado.
        2. estado: Booleano que muestra si un estado está activo o no.
        3. beneficioso: Un estado beneficioso es aquel que cuando una característica beneficiosa supera el valor de cambio de su transición es activada, mientras que si es una característica no beneficiosa es la que supera el valor si el estado es beneficioso se desactivaría.
    - Referencias a otros conceptos y su justificación:
        1. Modificado: Es la referencia entre los estados y las transiciones que modifican el estado, un estado puede ser modificado por 1 o n transiciones, La necesidad de crear esta relación es simplemente para poder tener un estado que sea activado o desactivado por varias transiciones, o lo que es lo mismo que dependa de varias características.
    - Restricciones:
        1. El atributo nombre no debe estar vacío.
- **Efectos**
    - Descripción: Los efectos son las modificaciones de características que son producidas por una determinada acción.
    - Atributos:
        1. nombre: String que almacenará el nombre del efecto.
        2. valorModifica: Entero que almacena el valor que una acción modificaría en una característica.
    - Referencias a otros conceptos y su justificación:
        1. modifica: Es la referencia entre efectos y características que son modificadas por el efecto, un efecto puede modificar a una única característica. La necesidad de crear esta relación es para controlar que un efecto no pueda afectar a dos características diferentes y poder dibujar el efecto en Eugenia.
        2. Causada: Es la referencia entre efectos y acciones que producen el efecto, un efecto puede ser causado por una única acción. La necesidad de crear esta relación es controlar que un efecto solo sea creado por una única acción y poder representar un efecto en el Eugenia.
    - Restricciones:
        1. El atributo nombre no debe estar vacío.

2. El atributo valorModifica no puede estar vacío.

- **Transiciones**
    - Descripción: Las transiciones son los encargados de activar o desactivar los estados de la mascota dependiendo de los valores que tengan las características de nuestra mascota en un momento determinado.
    - Atributos:
        1. nombre: String que almacenará el nombre de la acción.
        2. valorCambio: Valor que al ser superado cambia el estado de desactivado a activado o al igualarlo o no superarlo lo cambia a desactivado si estaba activo.
    - Referencias a otros conceptos y su justificación:
        1. producida: Es la referencia entre los efectos y las características que han producido la transición, una transición es producida por una única característica. La necesidad de crear esta relación es para controlar que una transición solo sea producida por una característica y además poder representar la transición en Eugenia.
    - Restricciones:
        1. El atributo nombre no debe estar vacío.
        2. El atributo valorCambio no debe estar vacío, debe ser mayor o igual y debe ser menor que el valorMax de características.

Sintaxis concreta

Una vez en el entorno de Eugenia, puedes comenzar a crear un modelo en base al metamodelo que nosotros hemos creado, dentro de la interfaz deberás crear un archivo de tipo file dentro de un proyecto, allí podremos crear dos tipos de elementos, nodos y enlaces:

Los nodos serían los siguientes.

Acciones: Serian las opciones que nos darían en el juego para que realice nuestra mascota, las acciones pueden causar de 1 a n efectos siempre cada uno de estos efectos a distintas características, es decir, una acción no puede crear dos efectos que afecten a la misma característica, las acciones solo tienen el atributo nombre que es obligatorio que este relleno.

Características: Las características son las distintas cualidades que tiene la mascota, del valor actual de estas características dependerán los estados activos o no activos de la mascota, para cambiar un estado las características crean de 1 a n transiciones que se encargarían de que cuando el valor de la característica supere el valor de cambio de la transición se active o desactive un estado. Los atributos que hay que rellenar en el entorno son los siguientes: Nombre, que deberá estar relleno, valor, es el valor actual de la característica, no puede ser ni mayor ni igual que el valorMax ni tampoco menor a 0, como el nombre también debe estar relleno, el siguiente atributo es valorMax que es el valor más alto que puede tener una característica, no puede ser menor o igual que 0 y debe estar relleno, y por último, su ultimo atributo es beneficioso, que deberá ponerse a true si la característica es buena (Felicidad…), o a false si la característica es mala (Tristeza, cansancio…). Las Características se subdividirán en dos tipos de características:

- Emocionales: Representan cualidades mentales o psicológicas de la mascota.

- Físicas: Representan cualidades del cuerpo de la mascota.

Estados: Los estados son la situación actual en la que se encuentra la mascota, los estados pueden estar activados estado=true, o desactivados, estado=false, aunque dependen de las características por lo que este atributo da igual como este inicializado, después tienen el nombre, que debe estar relleno, por ultimo estaría el atributo beneficioso que sería igual que el mismo campo beneficioso de características, true si el estado es bueno y false si es malo.

Los estados también estarán subdivididos en dos:

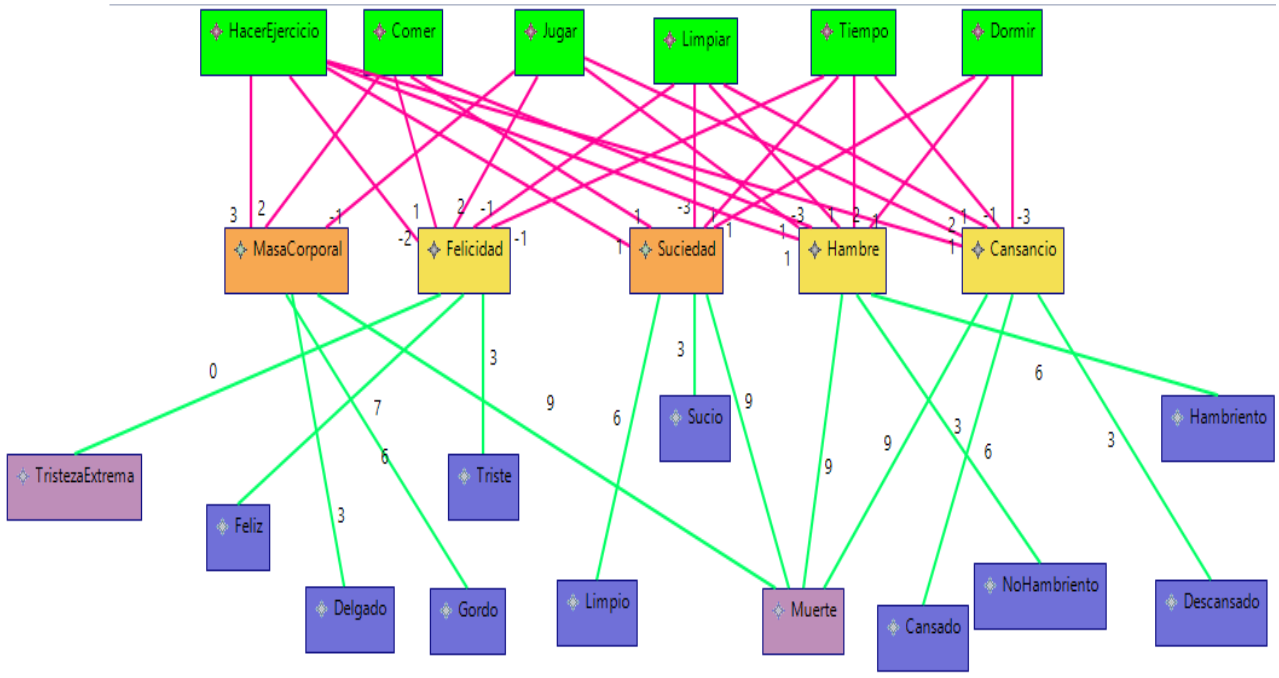
- Normales: Son los estados que no causan la muerte de la mascota cuando están activados o desactivados, pueden ser beneficiosos o no beneficiosos,
- Críticos: Son los estados que causan la muerte de la mascota, en el caso de los beneficiosos al desactivarse y en el caso de los no beneficiosos al activarse.

Los enlaces serían los siguientes:

Efectos: Se dibujarían entre una acción y la característica que quieres que se modifique cuando se realiza una acción, los efectos son las modificaciones de las características que causan las acciones, una acción puede crear muchos efectos, pero no dos sobre la misma característica, son representados mediante una línea verde entre una acción y una característica y en la unión se pone el valor que modificaría en la característica. Tiene dos atributos, el valorModifica y el nombre, ambos deben estar rellenos.

Transiciones: Se dibujarían entre una característica y un estado, una transición es lo que causa que cambie un estado entre activado y desactivado cuando se cumple que el valor de la característica supera al atributo valor cambio de la transición, una característica puede causar múltiples transiciones pero una transición solo puede ser causada por una característica, del mismo modo, una transición solo puede afectar a un estado, mientras que un estado puede ser modificado por múltiples transiciones, los atributos de una transición son el nombre y el valor cambio, ambos campos deben estar rellenos y además el valor de cambio debe ser menor que el valorMax de la característica.

Ejemplo de modelo



Generación de Código

La generación de código nos generara un gran número de clases que dividiré en dos apartados:

Aplicación Mascota: En este apartado incluiré las clases que se generan según el metamodelo de Pouex, en este apartado incluiré las clases que afectan a la mascota o están incluidos en la mascota, las clases generadas serán las siguientes:

- Mascota
- Acciones
- Características
- Físicas
- Emocionales
- Estados
- Normales
- Críticos
- Efectos
- Transiciones

Interfaz Gráfica Mascota: Es la interfaz que hemos creado para la aplicación, no está incluida en el metamodelo y es una ampliación opcional del proyecto, como anteriormente no explique cuál era su función lo explicaré a continuación ya que las del apartado anterior están explicadas mas arriba:

- Botón Acción: Crea un botón con una acción relacionada y un listener esperando, de forma que cuando es pulsado se ejecuta el método realizarAccion de la acción que se encuentre relacionada al botón.
- Interfaz: Crea la ventana de la interfaz del juego con las propiedades, los botones de

guardado y las acciones del juego, cambia la acción tiempo por pasar tiempo, el tiempo se realizaría cada 15 segundos si se ha creado en el modelo la acción tiempo, si no se ejecutaría la acción.

- InterfazMuerte: Muestra una ventana cuando un estado crítico beneficiosos esta desactivado o un efecto no beneficioso se activa en la que se muestra que el juego ha finalizado, esta interfaz cerraría el programa.

No pondremos el código de las clases ya que sería demasiado largo y realmente no vemos demasiada utilidad a ponerlo todo por lo que simplemente pondremos el código que genera la inicialización del programa cogiendo los datos del .xmi o modelo generado con el Eugenia.

Código Acceleo:

```
	public static Mascota crearMascota() {
		// Creacion de Mascota
		Mascota M = new Mascota("[aMascota.nombre/]");

		// Creacion de estados
		[for (a : Estados | aMascota.estar)]
		Estados [a.nombre/] = new [a.eClass().name/]("[a.nombre/]",
[a.beneficioso/], [a.estado/]);
		M.anhadirEstado([a.nombre/]);
		[/for]



		// Creacion de Caracteristicas
		[for (a : Caracteristicas | aMascota.poseer)]
		Caracteristicas [a.nombre/] = new [a.eClass().name/]("[a.nombre/]",
[a.beneficioso/], [a.valor/], [a.valorMax/]);
			[for (b : Transiciones | a.causar)]
			Transiciones [b.nombre/] = new Transiciones("[b.nombre/]",
[b.valorCambio/], [b.modificar.nombre/]);
			[a.nombre/].anhadirTransicion([b.nombre/]);
			[/for]
		M.anhadirCaracteristica([a.nombre/]);
		[/for]

		// Creacion de Acciones
		[for (a : Acciones | aMascota.realizar)]
		Acciones [a.nombre/] = new Acciones("[a.nombre/]");
			[for (b : Efectos | a.causa)]
			Efectos [b.nombre/] = new Efectos("[b.nombre/]", [b.valorModifica/],
[b.modifica.nombre/]);
			[a.nombre/].anhadirEfecto([b.nombre/]);
			[/for]
		M.anhadirAccion([a.nombre/]);
		[/for]

		M.igualarEstadosCaracteristicas();

		return M;
	}
```

Para ejecutar este código todas las clases generadas deben meterse dentro de la carpeta src de un proyecto java.

Herramientas de desarrollo

Para la realización de este proyecto usamos eclipse en su versión Kepler, y utilizamos varios software adicionales de eclipse como son los siguientes:

Eclipse Modelling Tools ver 2.0.1: Lo usamos para generar los modelos de nuestro sistema.

OCL ver 3.3.1: Utilizado para poder crear restricciones ocl para nuestro sistema.

Acceleo ver 3.4.2: Utilizado para la generación del código java de la aplicación en mase a un modelo de entrada.

Eugenia 1.1: Utilizado para poder generar modelos en base a nuestro metamodelo de forma gráfica.