

Autómatas y animaciones (II)

Créditos

- » **Autor:** Hugo Ruscitti
- » **Fecha:** 30 de Julio del 2005

Introducción

Un personaje de video juego suele realizar varios movimientos simples como: saltar, correr, disparar etc.

En algunos juegos la cantidad de movimientos por personaje es numerosa y suele combinarse. Por ejemplo, en el videojuego Street Fighter el jugador puede "golpear mientras salta", o "patear agachado en el suelo".

En este último caso, programar el comportamiento de un personaje se hace más difícil a medida que crece el número de acciones involucradas.

En este artículo veremos como gestionar un conjunto de movimientos mediante autómatas finitos. Al final del artículo se presenta una pequeña demostración de un video juego de peleas.

Estados

Un estado representa el comportamiento del personaje en un momento dado. Cada estado reacciona de manera diferente a los eventos (como la pulsación de una tecla) y está asociado a una animación diferente:



En este caso, el personaje de la figura cuenta con el estado PARADO, ya que tiene una animación asociada y reacciona de manera diferente ante un evento como "golpear", en comparación a otro estado como SALTA.

Autómata

Un autómata finito es un modelo lógico asociado a una serie de propiedades, entre ellas, un conjunto de estados, un alfabeto y una relación de transición.

Pero para nuestra aplicación no será necesario profundizar en la definición formal de autómatas, en su lugar optamos por presentar el concepto de manera informal.

Interpretación informal del autómata:

Podemos imaginar a un autómata como una máquina que se encuentra en un estado particular, y que ante

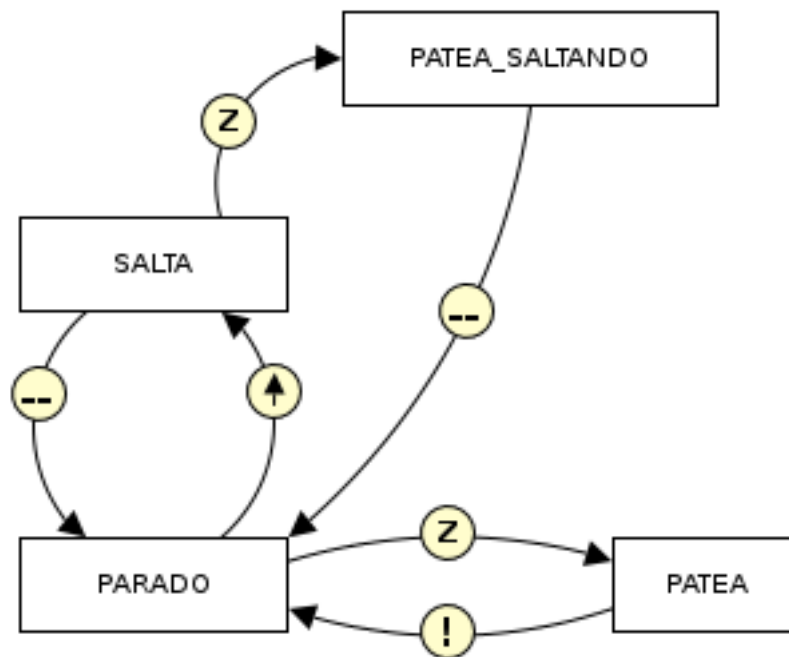
determinados eventos cambia de un estado a otro.

Nuestro personaje funcionará como un autómata:



Cuando comienza el juego estamos en posición de combate (un estado), si pulsamos hacia arriba (un evento) comenzamos a saltar. Saltar será nuestro nuevo estado, y en él tendremos una serie de posibilidades nuevas como realizar una "patada voladora".

Este criterio se puede apreciar fácilmente mediante un diagrama de transiciones:



Referencia de eventos

- | | |
|------------------------|--------------------|
| ⓘ Termina la animacion | -- Toca el suelo |
| ↑ Pulsa la tecla Subir | Z Pulsa la tecla Z |

diagrama de transiciones simple

De la figura podemos deducir algunas características:

- » El autómata no puede estar en mas de un estado al mismo tiempo.
- » Los eventos relacionan a los estados entre sí.
- » No todos los eventos implican un cambio de estado, para este ejemplo un evento como "agachar" no tiene sentido mientras el personaje se encuentra en estado "salta".

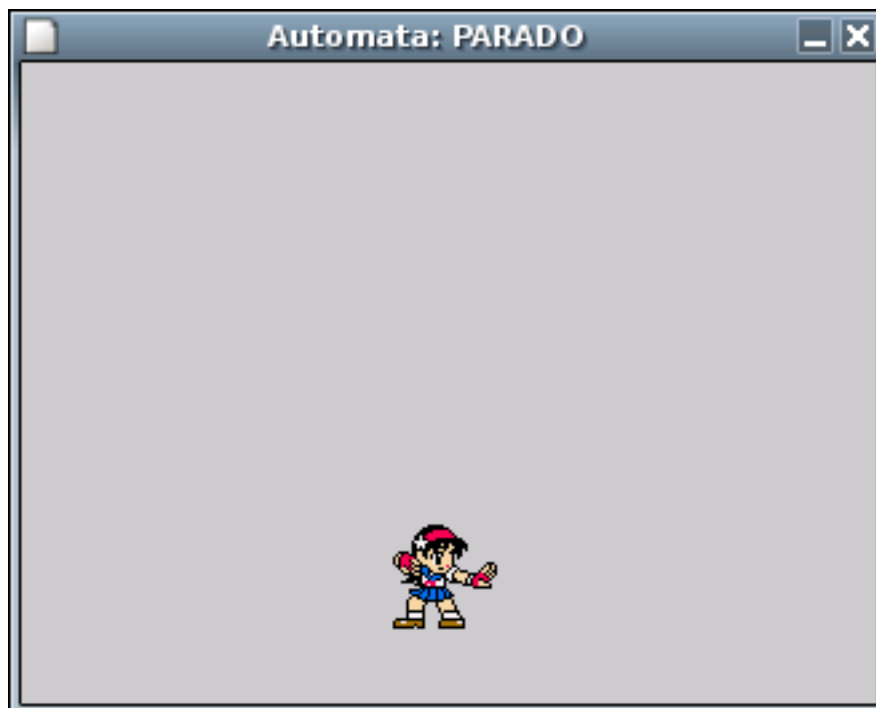
Implementando un gestor de estados:

Existen varias formas de implementar un gestor de estados partiendo de un autómata. Si bien el modelo original será idéntico, cada implementación difiere en eficiencia y complejidad.

Aquí, como en el resto de los artículos, optamos por un criterio simple y fácil de implementar, el objetivo de los ejemplos es brindar una base de prueba para interpretar. Seguramente encontrará formas más eficientes de implementar este programa utilizando punteros a funciones o polimorfismo.

Ejemplo:

A continuación podrá descargar el ejemplo del artículo.



El programa de ejemplo en GNU/Linux

- » Versión ejecutable para Windows
- » Código fuente en lenguaje C para GNU/Linux
- » Ver el código fuente on-line
- » Bajar el diagrama de transiciones en formato .dia

En el programa podemos manejar al personaje con los direccionales del teclado y mediante las teclas 'x' y 'z' podemos "patear" y "golpear con el puño" respectivamente.

El título de la ventana nos indica el estado actual del autómata.

El personaje almacena su estado actual junto con el resto de sus propiedades dentro de la estructura actor:

```
enum estado { PARADO, SALTA, GOLPEA [...] };

struct actor
{
  enum estado;
  int x;
  int y;
  [...]
}
```

El manejo general del personaje consiste en llamar de manera periódica a la función:

```
void actor_actualizar (struct actor * obj)
```

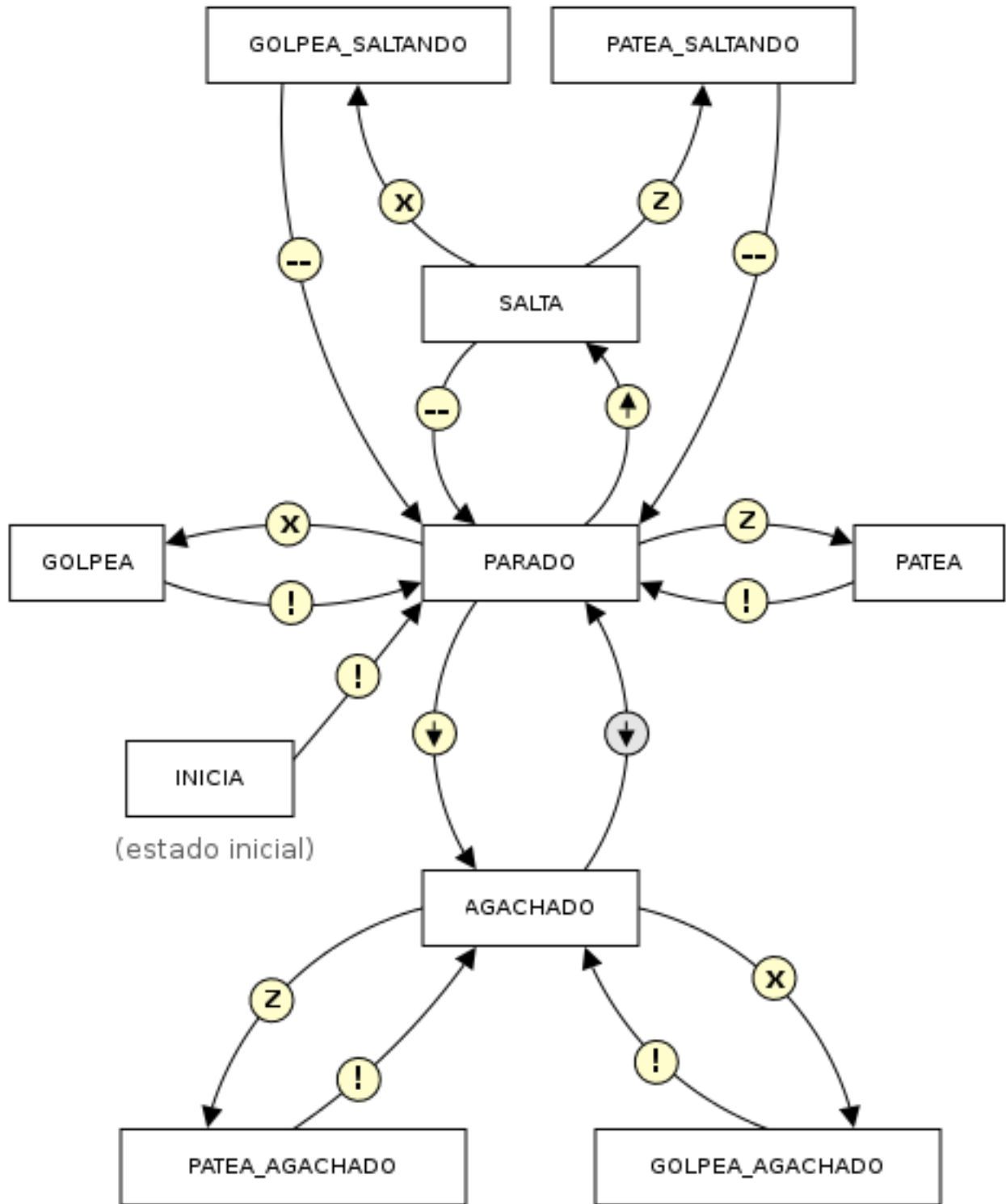
que selecciona, en base al estado del actor, que función se debe procesar para representar el comportamiento en ese instante.

Así, cada estado se encuentra asociado a una función independiente que recibe los datos del personaje y las teclas pulsadas:









```
void actor_estado_parado (struct * actor obj, Uint8 * key);
void actor_estado_camina (struct * actor obj, Uint8 * key);
void actor_estado_agachado (struct * actor obj, Uint8 * key);
[...]
```

Diagrama de transiciones:

El siguiente gráfico es una versión reducida del diagrama de transiciones utilizado en el ejemplo:



Referencia

-  Termina la animacion
-  Toca el suelo
-  Pulsa la tecla Izquierda
-  Pulsa la tecla Derecha
-  Pulsa la tecla Subir
-  Pulsa la tecla Bajar
-  Pulsa la tecla X
-  Pulsa la tecla Z
-  No se pulsa la tecla

Conclusión:

La mayor parte del trabajo dedicado a esta técnica consiste en diseñar un autómata que cubra nuestros requerimientos; a partir de ahí, la implementación posterior es muy sencilla y fácil de modificar.

Licencia

Se permite la copia, modificación y distribución de este artículo sólo bajo los términos de la Licencia Creative Commons.

Este documento ha sido generado automáticamente a partir del archivo 'automatas.xml' el Mon Jan 19 09:32:57 2009

La versión mas reciente de este documento se almacena en www.losersjuegos.com.ar. Visitenos para obtener mas recursos y actualizaciones.