II Jornadas de proyectos de Software Libre

Racing Car

El paso de una idea a un videojuego *libre*.

Miguel Ángel Díaz López

Alumno de 4º curso de Ingenería Informática.
Universidad de Granada.

21 de Octubre del 2010

1 Primeras Preguntas

2 Arquitectura de Racing Car
- Arquitectura básica: colisiones
  - Colisiones: partículas-escenario
  - Colisiones: partículas-partículas
- Arquitectura básica: máquina de estados
  - Máquina de estados: escenario
  - Máquina de estados: partículas

3 Software Libre

¿Qué es *Racing Car*?

Racing Car ha sido una apuesta para el proyecto de Informática Gráfica *(3º.OP)* impartida en la ETSIIT.

1. Inicialmente consistía en una práctica de modelado en 3D, donde coches *replicados* evitaban colisiones y seguían un simple AFD.
2. Finalmente evolucionó a un **videojuego** en 3D donde coches de diferente épocas y de miniatura *luchan* por ganar una carrera; utilizando una casa como escenario; y convirtiéndose en **software libre**.

¿Cómo se ha desarrollado?

Para ello hemos utilizado Qt, consistente en un sistema multiplataforma de aplicaciones C++; a la cual se le pueden agregar *bindings*, en nuestro caso OpenGL.

¿Ha influido el concepto de SL en la plataforma elegida?

En un principio, posiblemente mi respuesta fuese negativa; sin embargo la combinación de la apariencia del entorno de desarrollo junto la licencia GPL, fueron características determinantes.

Finalmente, la decisión acertada de utilizar software **no** privativo ayudo a la liberación de *Racing Car*.

¿Por qué Qt?

1. Multiplataforma.
2. Utiliza la licencia: GPL v2/v3, *entre otras*.
3. Amplio Respaldo: VLC media player, VirtualBox, Skype, KDE, Autodesk...
4. Cuestiones personales: apariencia (entorno de desarrollo amigable), facilidad de uso, herramientas...

¿Cómo se desarrolla un videojuego?

Según mi experiencia durante el periodo de creación y desarrollo de Racing Car:

1. **Concienciarse** que no es un abrir y cerrar de ojos.
2. Buscar una **idea** a desarrollar.
3. Centrarse en el conjunto de **herramientas** que se van a utilizar: buscar, constrastar y seleccionar.
4. Si se conocen ya los lenguajes de programación elegidos y las herramientas, **elaborar** el videojuego. Tarea, más frustrante y gloriosa a la vez.
5. **Mantenimiento**.

Vídeo Racing Car

Aquí podemos ver un vídeo a modo ejemplo del Videojuego, no obstante, se ve poco fuido y baja calidad debido a la grabación del escritorio y la compresión.

Figura: Entrada de la casa.

Figura: En el salón del escenario.

Figura: Entrando a la cocina.

Pulsar para ver vídeo: Racing Car

Arquitectura básica: colisiones

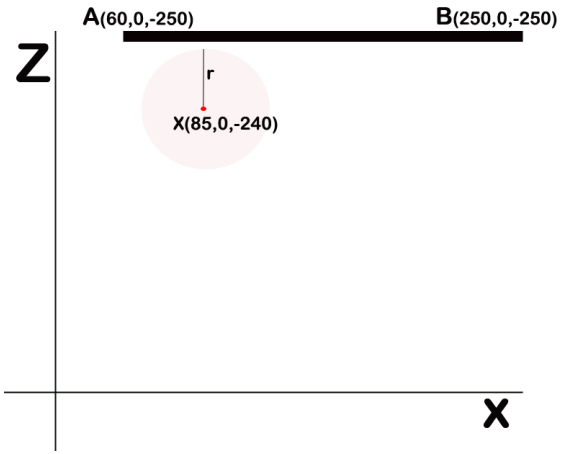
Yo creo que uno de los aspectos básicos de la arquitectura es la interacción de las *partículas* con el entorno y con ellas mismas, es decir, la consciencia de otros *seres* y objetos.
"Dejamos el modelo egocéntrico y *abrimos los ojos* de las *partículas*."

¿Cómo despertamos dicha *consciencia* del mundo?

Colisiones: partículas-escenario

En el caso de las *partículas* con el escenario necesitamos aplicar los conceptos:

1. **Ecuación de la recta** que pasa por dos puntos.
2. La **distancia** que existe entre un punto y una recta.
3. **Caja frontera** de una *partícula*, en Racing Car, el radio mínimo que encierra a dicha *partícula* en una *esfera*.

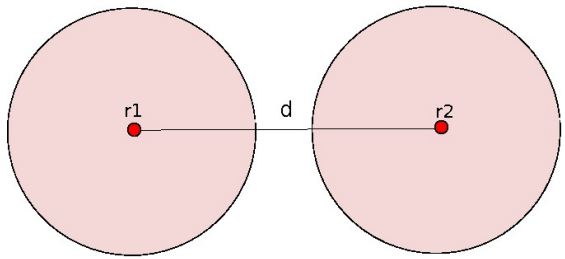


¿Solucionado?
¿Acaso no puede prolongarse una recta hasta el **infinito**?

Colisiones: partículas-partículas

Para la colisión entre una partícula animada y otras el algoritmo utilizado en **Racing Car** ha sido el siguiente:

1. **Comprobar** si existe o no posibles colisiones con el resto de partículas animadas.
2. Realizamos comprobaciones a pequeñas distancias de nuestra posición, **predecir** futuras colisiones.
3. En caso que **exista** una futura colisión: dentro de las partículas colisionadas, buscamos la partícula más adelantada y damos prioridad máxima, frentre las demás implicadas, para que *salga* del estado de colisión. **Sistema de preferencias**.
4. En caso de no existir colisiones, continuamos movimiento.



Arquitectura básica: máquina de estados

En **Racing Car**, y quizas en la mayoria de videojuegos, surgen tambien este otro tipo de dudas que hay que resolver :

1. ¿Cómo el ordenador **controla** las diferentes partículas?
2. ¿Son diferentes, una partículas de otras?
3. ¿Cómo determinan el camino a seguir: *adelantamientos, repostaje, frenadas*...?
4. ...

La solución tomada, en el caso de *Racing Car*, ha sido emplear Autómatas Finitos No Deterministas, **AFND**, como técnica de control racional en el contexto de la Inteligencia Artificial.

¿Qué es un FSA o una Máquina de Estados Finitos (FSM)?
Son modelos de comportamiento de un sistema o un objeto complejo, con un número limitado de modos o condiciones predefinidos, donde existen transiciones de modo.

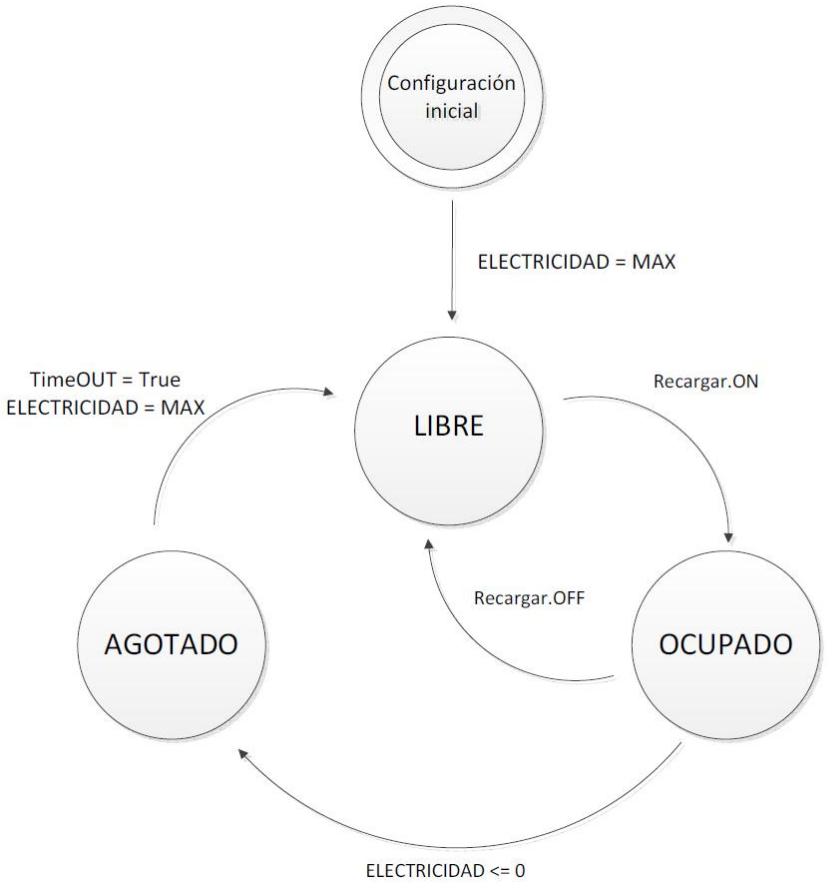
Las Máquinas de Estados Finitos, estan formadas por 4 elementos principalmente:

1. Estados.
2. Transiciones de estado.
3. Reglas o condiciones.
4. Eventos de entrada.

En *Racing Car* se han utilizado dos máquinas de estados: en el **escenario** y en las **partículas**.

Máquina de estados: escenario

Son los diferentes modos de comportamiento que adopta el sitema escenario al interactuar con las partículas,



Máquina de estados: partículas

Aquí podemos diferenciar entre las partículas controladas por el ordenador y las controladas por el ser humano, haciendo esta distinción, obtenemos dos máquinas de estados.

1. Máquina de estados de partículas controladas por el **ser humano**.
2. Máquina de estados de partículas controladas por el **ordenador**.

Máquina de estados: partículas controladas por el ser humano

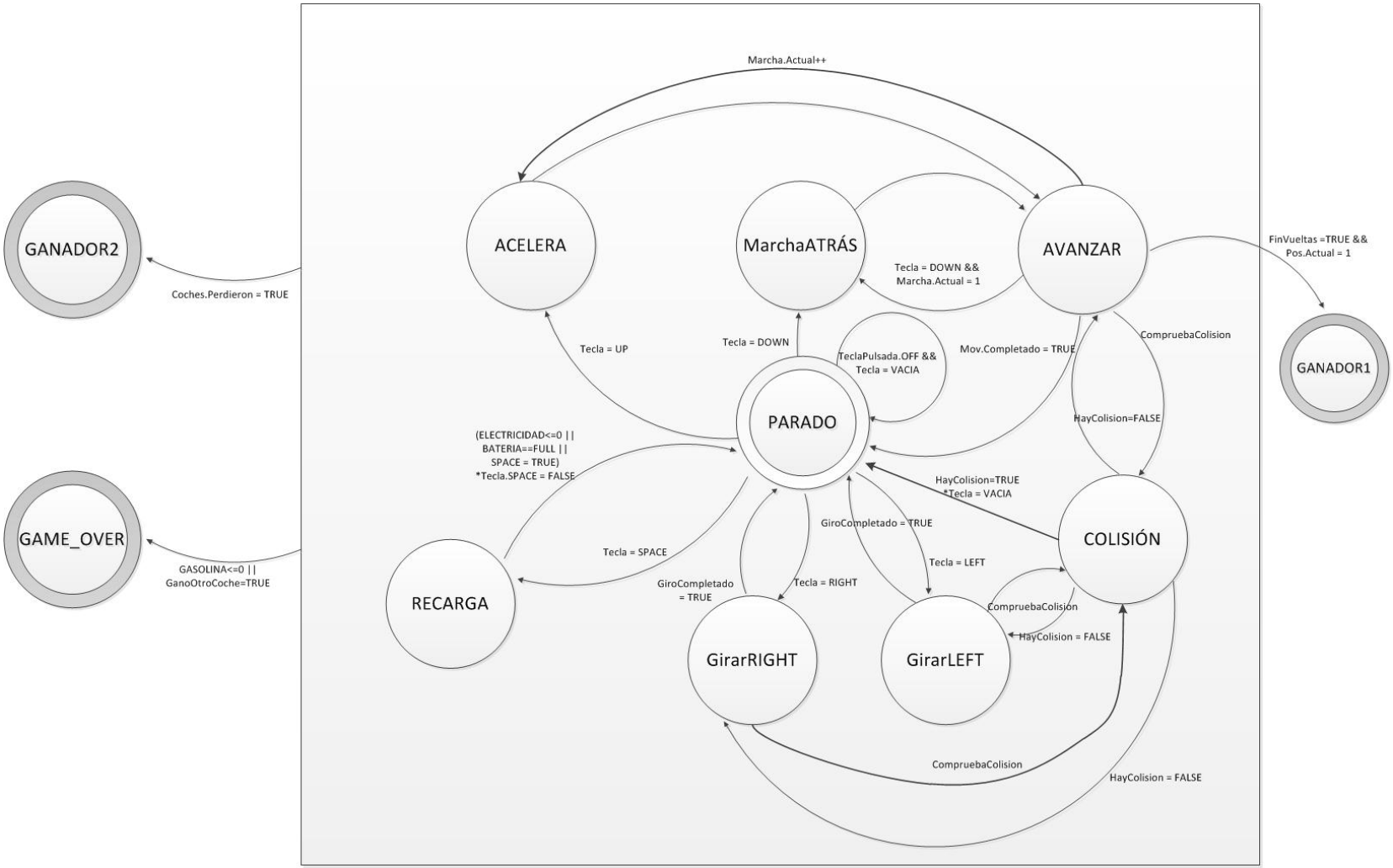
Estados específicos de las partículas controladas por el **ser humano**:

1. Acelerar
2. Marcha Atrás
3. Girar Left
4. Girar Right
5. Parado
6. Colisión

Estados del AFND **comunes**:

1. Avanzar
2. Recargando
3. Ganador
4. Game Over

Máquina de estados: partículas controladas por el ser humano



Máquina de estados: partículas controladas por el ordenador

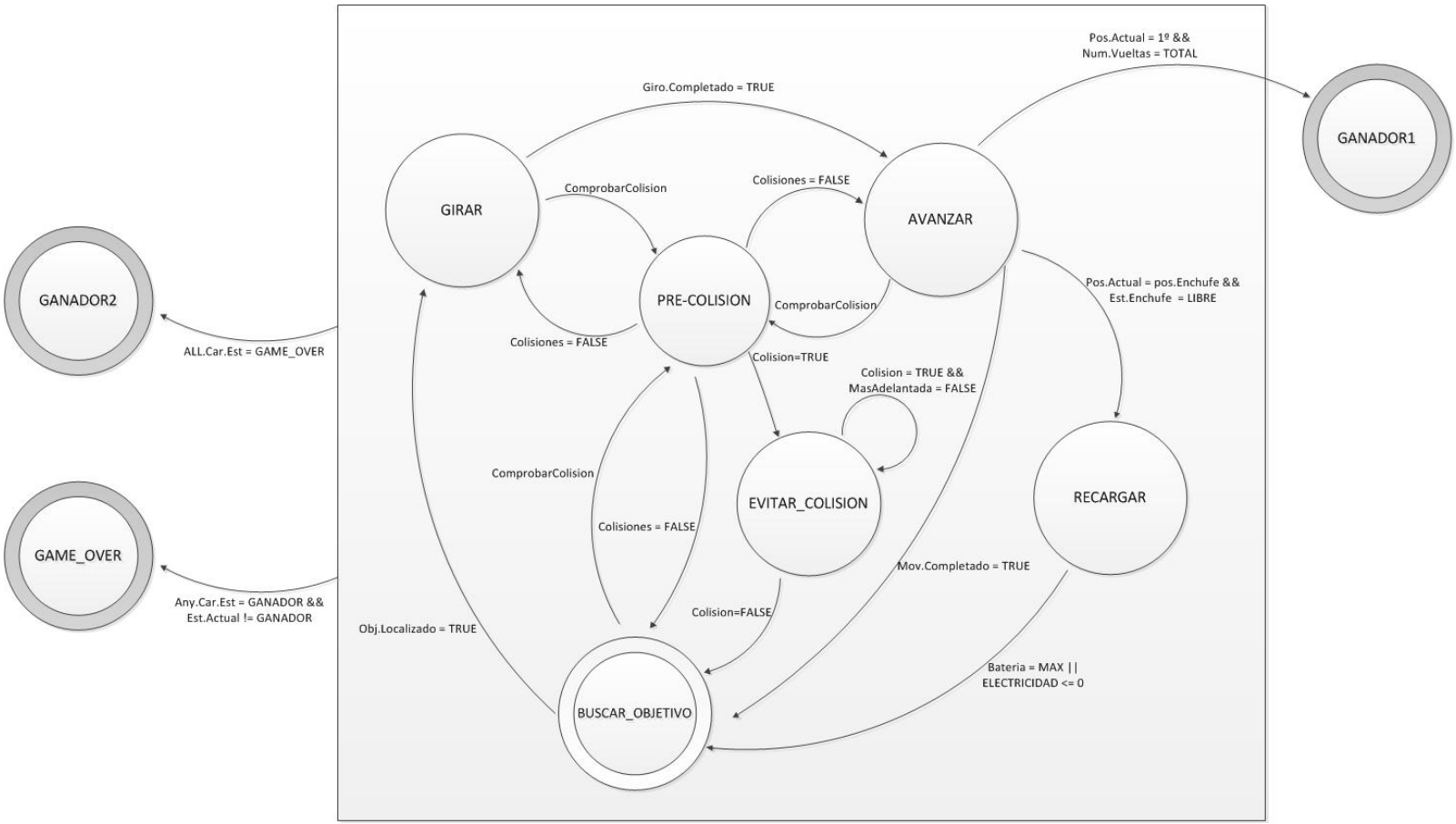
Estados específicos de las partículas controladas por **el ordenador**:

1. Buscar Objetivo
2. Girar
3. Evitar Colisión
4. Pre-Colisión

Estados del AFND **comunes**:

1. Avanzar
2. Recargando
3. Ganador
4. Game Over

Máquina de estados: partículas controladas por el ordenador



Software Libre

¿Por qué se ha liberado *Racing Car*? **¿Por qué liberar?**

1. **Acreditación** como autor.
2. **Acercarse** al mayor número de personas.
3. Generar nuevas posibilidades de **trabajo**.
4. Desarrollo de software *a priori* **inviable** económicamente.
5. **Mantenimiento** compartido.
6. **Colaborar** en el desarrollo de futuros softwares.
7. **...**

Podeís descargar una copia de **Racing Car** en:

Http://code.google.com/p/racing-car/

¿Qué posibles mejoras se podrían añadir a *Racing Car*?

1. **Optimización** de los recursos del sistema.
2. Mejoras en los **movimientos**: incluir giro durante la trayectoria.
3. Distintos niveles de **dificultad**.
4. Nuevos **escenarios**.
5. **Empaquetado** para diversas plataformas.
6. **...**