Arquitectura de videojuegos

Máster en Programación de videojuegos

Javier Arévalo

2º Trimestre
Curso 2013-2014

Arquitectura de videojuegos

Separación de Sistemas
Lógica, Input y Render


XX
ZZ
XY
ZW

Ejecucion del juego

- Hasta ahora siempre hemos estado hablando de un CurrentState.Run() monolítico

```
CurrentState = null
WantedState = InitialLoad

while (true):
  if (CurrentState != WantedState):
    CurrentState.Deactivate().Unload().Destroy()
    if (WantedState == null):
      break
    CurrentState = WantedState
    CurrentState.Initialize().Load().Activate()
    WantedState = null

  CurrentState.Run()
```

Separación de Sistemas

- Detrás de esa llamada se tiene que gestionar toda la ejecución del juego
- El juego esta dividido lógicamente en diferentes subsistemas
- Para evitar dependencias demasiado fuertes, esos subsistemas deben estructurarse de forma aislada
- Ventajas en sistemas modernos multi-core

Subsistemas

- Lógica
  - Ejecución de comportamientos, movimiento, física, Inteligencia Artificial, colisiones, etc.
- Control / Input
  - Captura de datos generados de forma externa al juego: sobre todo, entrada por parte del jugador
- Rendering
  - Gráficos, sonido, en general feedback al jugador

Subsistemas (2)

- Red
  - Gestión de las comunicaciones entre jugadores y/o con un servidor
- Datos
  - Carga de datos desde archivos, disco, server
- Otros
  - Achievements, estadisticas, métricas
  - ...

Subsistemas

- Nos quedamos con los tres grandes

```
while (true):
  CurrentState.Run()
```

```
while (true):
  CurrentState.Input()
  CurrentState.Logic()
  CurrentState.Render()
```

```
previousTime = GetTime()
while (true):
  currentTime  = GetTime()
  elapsed = currentTime - previousTime
  previousTime = currentTime

  CurrentState.Input(elapsed)
  CurrentState.Logic(elapsed)
  CurrentState.Render(elapsed)
```

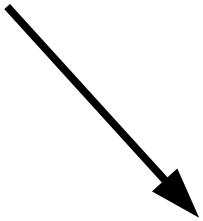
Importante

- El orden importa
  - A varios niveles, como veremos mas tarde
- Primero recogemos lo que hace el jugador
- Después actuamos en consecuencia
- Y por ultimo, lo pintamos

De aquel prototipo...

```
if (keyboard[37]) playerX--;
if (keyboard[39]) playerX++;
if (keyboard[38]) playerY--;
if (keyboard[40]) playerY++;

context.clearRect(0, 0, canvas.width, canvas.height);
context.fillStyle = "#0F0";
context.fillRect(playerX, playerY, 20, 20);
```



```
function Input(elapsed):
  playerDirectionX = 0;
  playerDirectionY = 0;
  if (keyboard[37]) playerDirectionX = -1;
  if (keyboard[39]) playerDirectionX = 1;
  if (keyboard[38]) playerDirectionY = -1;
  if (keyboard[40]) playerDirectionY = 1;

function Logic(elapsed):
  playerX += playerDirectionX*elapsed;
  playerY += playerDirectionY*elapsed;

function Render(elapsed):
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.fillStyle = "#0F0";
  context.fillRect(playerX, playerY, 20, 20);
```

Ventajas?

- Input:
  - Podemos configurar fácilmente diferentes dispositivos de control sin afectar a la lógica
  - Podemos grabar y reproducir automáticamente secuencias de input

Ventajas?

- Lógica:
  - Podemos pausar el juego simplemente no ejecutando la lógica
  - Podemos garantizar que la ejecución de la lógica no depende de ningún elemento externo al juego
    - Determinismo, una propiedad extremadamente útil
  - Podemos alterar el ritmo de ejecución de la lógica
    - Evitar aquella 'espiral de la muerte'

```
function Input(elapsed):
    playerDirectionX = 0;
    playerDirectionY = 0;
    if (keyboard[37]) playerDirectionX = -1;
    if (keyboard[39]) playerDirectionX = 1;
    if (keyboard[38]) playerDirectionY = -1;
    if (keyboard[40]) playerDirectionY = 1;
    if (keyboard[27]) paused = !paused;

function Logic(elapsed):
  if (!paused):
    playerX += playerDirectionX*elapsed;
    playerY += playerDirectionY*elapsed;

function Render(elapsed):
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.fillStyle = "#0F0";
  context.fillRect(playerX, playerY, 20, 20);
  if (paused):
    context.drawText("Game Paused");
```

Ventajas?

- Rendering:
  - Podemos modificar elementos de detalle y calidad del render sin afectar al resto del juego
  - Podemos eliminar esta parte para ejecución no interactiva
    - Servidores
    - Test automáticos
  - Podemos utilizar diferentes motores de render:
    - OpenGL vs DirectX, diferentes plataformas, etc

Input

- Normalmente nos referimos a los elementos de control del jugador
- En realidad, deberíamos hablar de cualquier tipo de evento externo al juego
  - Control
  - Red
  - Eventos del sistema
  - Callbacks de vuelta de librerías

Input

- Tener un sitio definido, garantizado y único en el que podemos ser interrumpidos por eventos externos
  - Aplicación pasa a segundo plano
  - Usuario ha realizado un logout
  - Controlador o red desconectados
  - Error de carga de disco
  - Ventana minimizada, contexto perdido, etc.

Input

- Grabación y reproducción de input
  - Registrar todos los eventos que puedan afectar a la ejecución de la lógica
  - Eventos de sistema y excepcionales normalmente no se envían a la lógica, aunque si pueden afectar su funcionamiento
    - Por ejemplo, desconexion → pausar el juego
  - Eventos de control pueden o no afectar a la lógica dependiendo de su destinatario
    - Nuestras interacciones con el submenu de configuración

Input

- Problema: lag en el control
- Input > Logica > Render > el jugador lo ve en su pantalla
- El tiempo que pasa desde pulsar un botón o mover el ratón, hasta ver el efecto, puede ser percibido
  - Las personas típicamente perciben lag alrededor de 1/10 de segundo

Input

- Solucion:
  - Hacer poll en un momento posterior, de aquellos elementos de control mas notables. P.ej. El ratón en un juego de primera persona
  - Los cursores de ratón hardware ya lo hacen por nosotros
- Input > Logica > Poll Ratón > Render
  - El giro de cámara esta tan actualizado como es físicamente posible
  - La lógica se ha ejecutado con el giro capturado en Input

Lógica

- Comportamiento del jugador
  - En función del input capturado
- Comportamiento de entidades
  - Percepción del entorno
  - Inteligencia Artificial
  - Tiempo de vida
  - Movimiento
  - Colisiones: ajuste y respuesta

Lógica

- El personaje del jugador es una entidad mas
  - El control hace las veces de 'Inteligencia Artificial'
  - Especialmente notable en juegos de estrategia
- Las entidades crean objetos propios para ser representadas
  - Objeto visual para rendering
  - Objeto emisor de sonidos
  - Etc

Lógica

- El 'mundo de juego' usara algún tipo de estructura espacial
  - Rejilla o tablero
  - Quadtree / octree
  - Grafo conectado
- Esa estructura esta adaptada a las necesidades de la lógica, no de rendering:
  - Percepción, física, IA.

Rendering

- Agrupamos aquí todos los sistemas que emiten información al jugador
  - Gráficos
  - Sonido
  - Interfaz de usuario
- Normalmente gráficos es el que va a tener necesidades mas amplias y complejas
  - Especialmente animación

Rendering

- Es habitual ejecutar render a ritmo y velocidad diferentes que la lógica
- El rendering tiene que ser correcto, pero sobre todo tiene que parecerlo
- A la lógica en si misma, le da igual ejecutar 10 ticks de golpe 1 vez por segundo, o 1 tick cada 1/10
- Pero el render tiene que dar sensación de fluidez

Rendering

- Animación
  - Parámetros que cambian de forma fluida a lo largo del tiempo
- Algunas animaciones responden directamente a comportamientos de la lógica
  - Ciclos de andar, disparar, etc
- Otras son estéticas y ocurren puramente dentro del rendering
  - Efectos visuales, movimiento de las olas del mar, etc

Rendering

- Su propia estructura espacial
  - Orientada a optimizar la detección de lo visible
- Evitar trabajo generado por elementos no visibles
  - Por cálculos de animacion
  - Por coste directo de pintado

Rendering

- Caso extremo: juego de tableros por turnos
- La lógica ejecuta un turno moviendo fichas de una casilla a otra
- El render muestra esos movimientos a lo largo de varios segundos:
  - Animando las fichas suavemente
  - Mostrando efectos visuales en los combates

Comunicación

- Los diferentes subsistemas tienen que comunicarse entre ellos
  - Cada juego y arquitectura son un mundo
- Casos tipicos:
  - Input genera un 'paquete' por cada frame de lógica que se deberá ejecutar
  - Rendering hace poll de elementos de input
  - Lógica crea y envía ordenes a objetos de rendering

Vuestro turno!

Preguntas?