

Guía de Flujo de Space Invaders

UPV

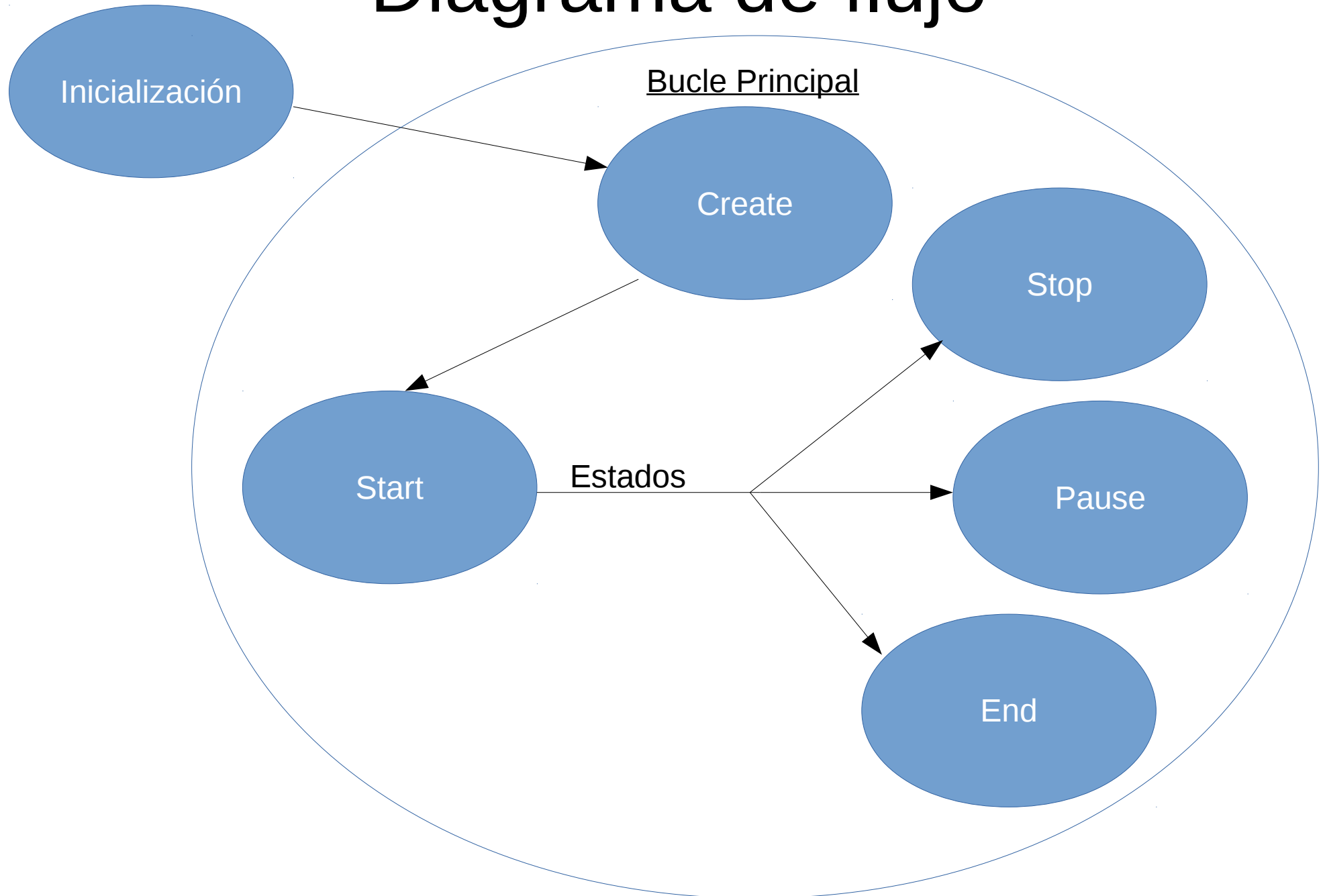


Jesús Alberto Alonso Nancloares, Eduardo Villa
Valdés

Índice

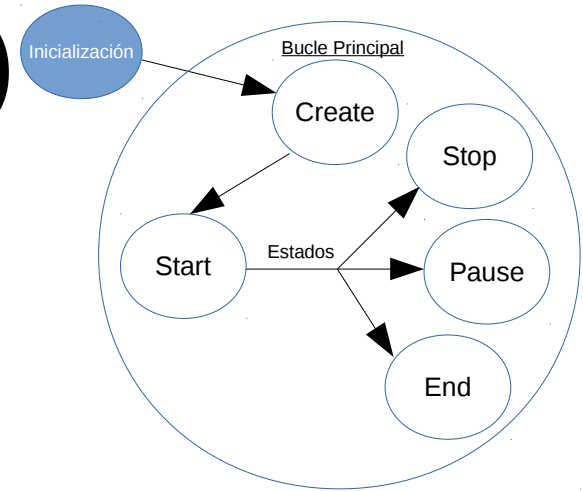
- Diagrama de flujo del juego
- Inicialización
- Game
- Bucle principal
- Objetos simulados

Diagrama de flujo



Inicialización (I)

- Carga de cadenas
- Carga de controles
- Creación de ventana y menús
- Carga de idioma
- Carga de niveles



Inicialización (II)

- Se realiza en `Main_SI_2013.cpp` y `SIGame.cpp`
- En `SIGame` se encuentran los métodos principales del juego, entre los que se encuentran la inicialización y el bucle principal.
- En `SIGame` se crean los directorios por defecto, los estados del juego, las transiciones entre estados, los sonidos y los temporizadores.

Carga de cadenas

- Inicializa las cadenas que usará el programa en adelante
- Se realiza en la sección Language del API. En el fichero UGKLanguage.cpp y UGKLanguageParser.cpp



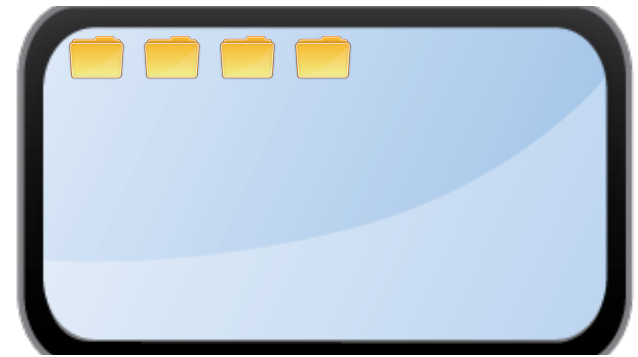
Carga de controles

- Inicializa los controladores de la ventana
- Se realiza en la sección Window del API, en los ficheros `GLWindow.cpp` y `MSWindow.cpp`

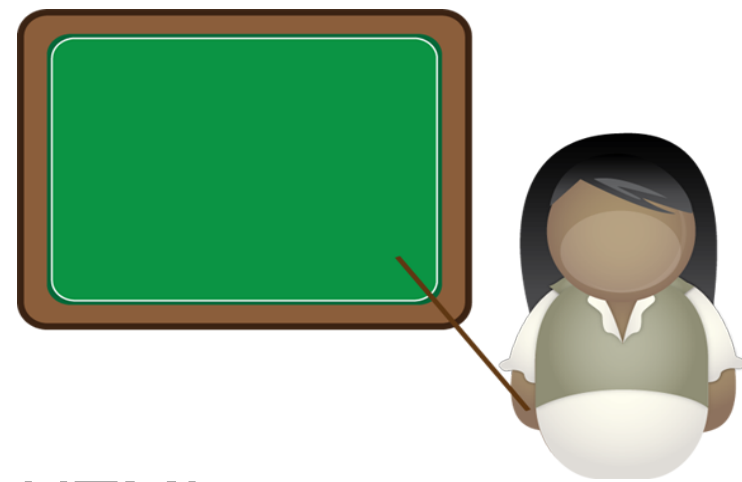


Creación de ventana y menús

- Carga la configuración de la ventana principal.
- Carga los menús de:
 - Space Invaders
 - Nivel
 - Ayuda
 - Idioma: francés, español, inglés y catalán
 - Empezar el juego
 - Salida

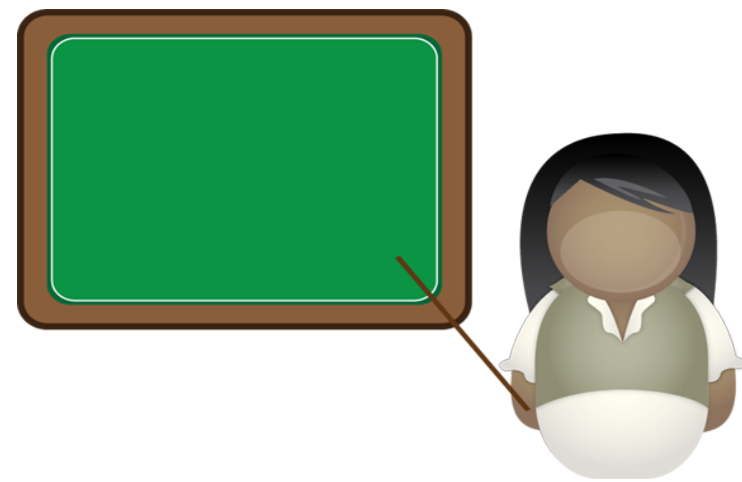


HTML Parser (I)



- HTML parser: interpreta los ficheros HTML proporcionados.
- Traduce los ficheros a la estructura interna de datos
- La ventaja de usar un lenguaje textual es la facilidad para crear nuevos contenidos como niveles, etc.
- El HTML se dedica a leer los tokens y analizar su validez.
- Primero carga inicialización, luego idioma y luego niveles de juego.

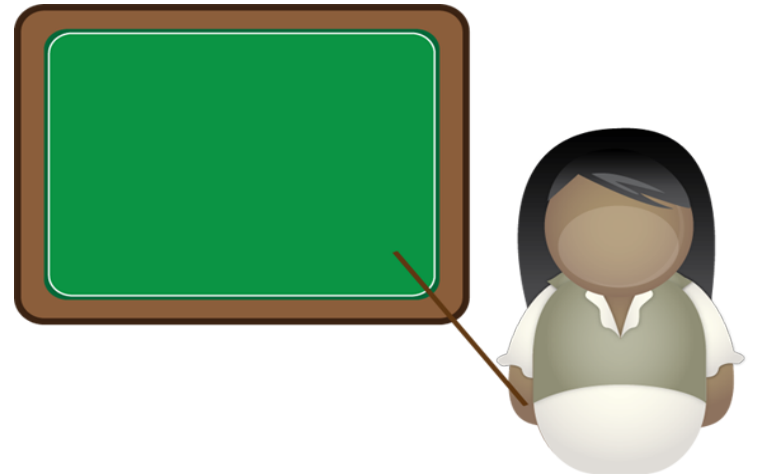
HTML Parser (II)



- Se compone de analizador léxico, sintáctico y semántico:
- Analizador léxico: define los tokens que puede encontrarse el parser. Acciones a realizar, según el token obtenido.
- Analizador sintáctico: interpretación de instrucciones o código que determine el comportamiento. Lenguaje de scripting.
- Analizador semántico: Generar estructura de datos interna del videojuego. Verifica que no existan errores.

Lenguaje (I)

- Lenguaje: carga en el lenguaje solicitado los mensajes predefinidos para el menú, el log y los mensajes de juego.

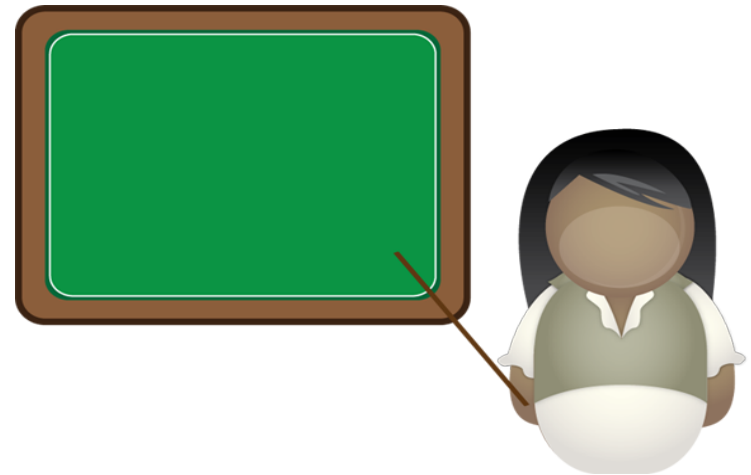


La internacionalización de un juego es una parte importante, ya que el objetivo es que este se use en distintos países

Se carga en el API Language y el archivo `application.cpp`

Lenguaje (II)

Para ello es imprescindible tener un paquete de idiomas con frases o textos predefinidos, ya que sería muy costoso escribir en el código lo que debe decir en cada idioma.



Lenguaje (III)

Carga de título, versión, tipo de archivo, en este caso el de lenguaje y lenguaje, en este caso español.

```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Space Invaders</TITLE>
```

```
    <VERSION>1.0.0</VERSION>
```

```
    <TYPE>Language</TYPE>
```

```
    <LANGUAGE>ES</LANGUAGE>
```

```
  </HEAD>
```

Lenguaje (IV)

Carga de los distintos mensajes predefinidos, estos se identifican por su ID cuando se solicita su contenido.

```
<BODY>
```

```
  <RESOURCE>
```

```
    <ID>1</ID>
```

```
    <CONTENT>Error de Internet</CONTENT>
```

```
  </RESOURCE>
```

```
  <RESOURCE>
```

```
    <ID>2</ID>
```

```
    <CONTENT>Información: conexión a Internet no se pueden  
    utilizar.</CONTENT>
```

```
  </RESOURCE>
```

```
  <RESOURCE>
```

```
    <ID>3</ID>
```

```
    <CONTENT>Advertencia</CONTENT>
```

```
  </RESOURCE>
```

Carga de niveles (I)

Carga de título, versión, tipo de archivo, en este caso el de nivel y nivel, en este caso el 0.

```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Space Invaders</TITLE>
```

```
    <VERSION>1.0.0</VERSION>
```

```
    <TYPE>Level</TYPE>
```

```
    <LEVEL>0</LEVEL>
```

```
  </HEAD>
```

Carga de niveles (II)

Carga de las distintas partes del nivel, con sus atributos.

```
<BODY>
```

```
  <CHARACTER>
```

```
    <NAME>GAME</NAME>
```

```
    <RENDERMODE>3D</RENDERMODE>
```

```
    <SIMULATIONMODE>DISCRETE</SIMULATIONMODE>
```

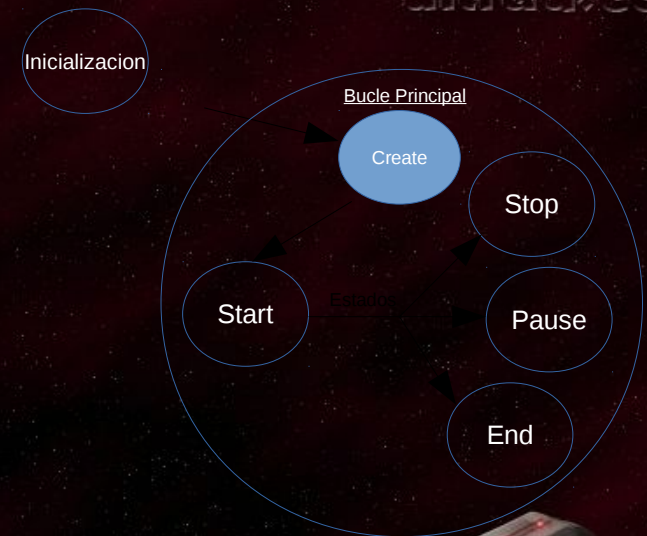
```
    <BOUNCE>0</BOUNCE>
```

```
    <TIMERENDER>16.67</TIMERENDER>
```

```
    <TIMEUPDATE>16.67</TIMEUPDATE>
```

```
  </CHARACTER>
```


Game (I)



La primera fase de todo programa es inicializar.

Esto se divide en las fases Initialize() e Init() que en un futuro se unificarán.

Game (I)

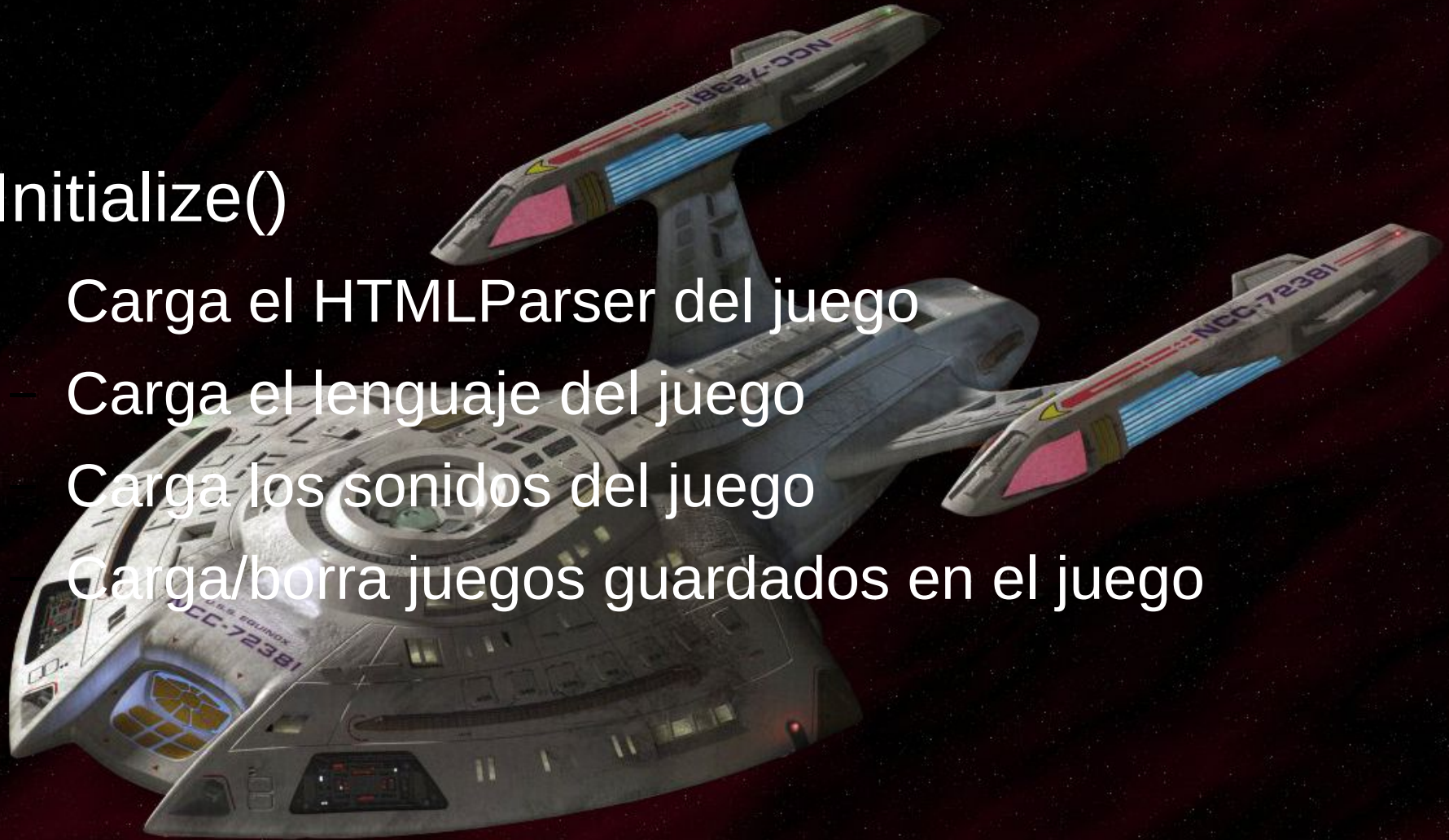
Initialize()

Carga el HTMLParser del juego

Carga el lenguaje del juego

Carga los sonidos del juego

Carga/borra juegos guardados en el juego



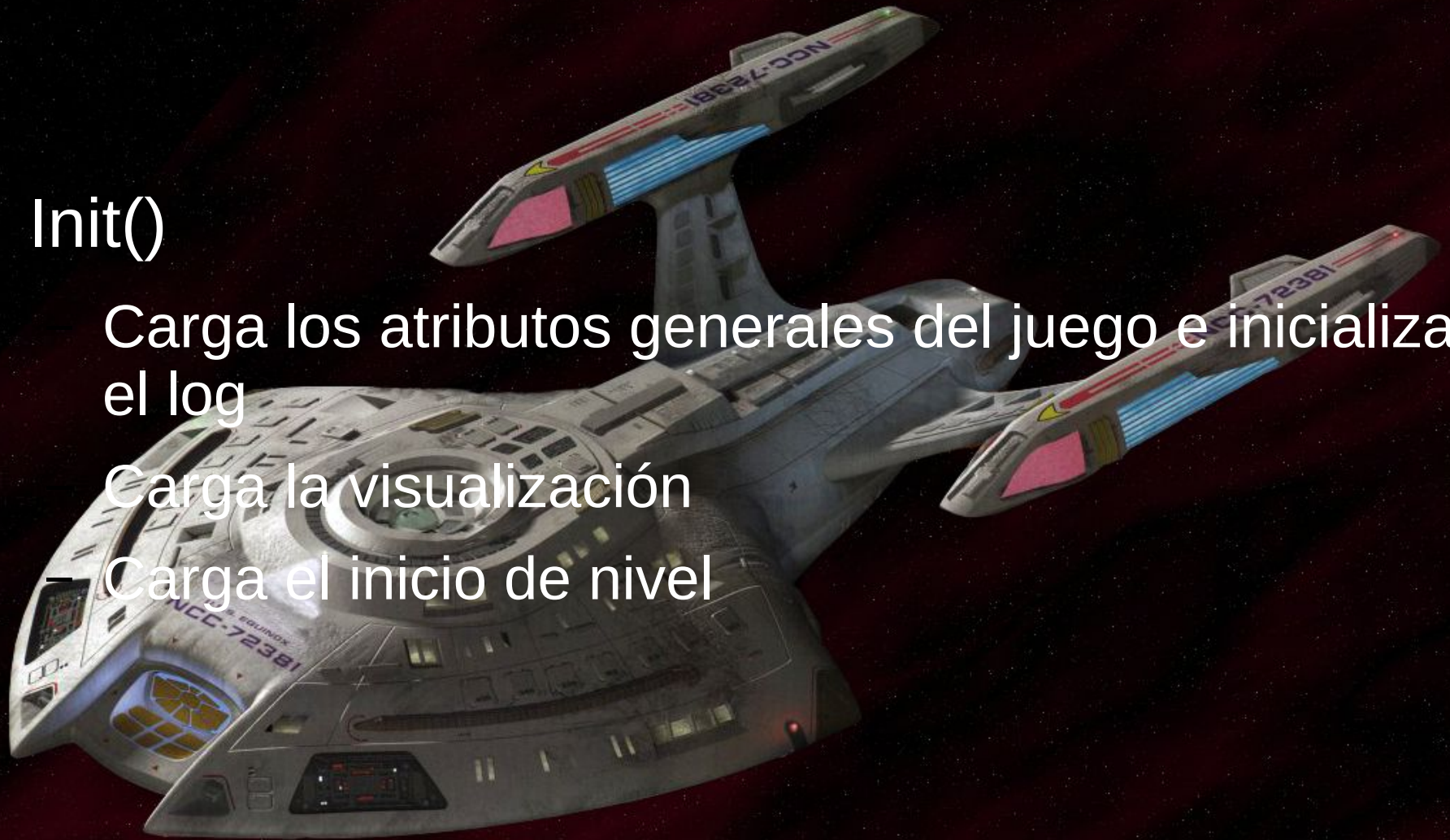
Game (II)

Init()

Carga los atributos generales del juego e inicializa el log

Carga la visualización

— Carga el inicio de nivel



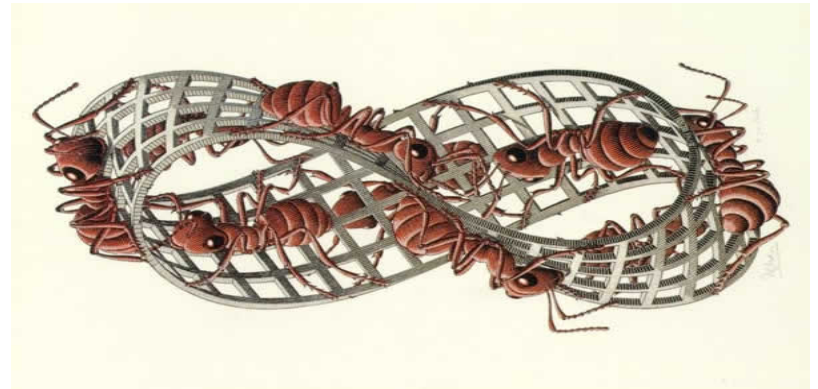
Bucle principal (I)

- El bucle principal actúa mientras reciba mensajes de RT-DESK
- El bucle principal se compone de:
 - APP_Init: crea la ventana de juego
 - APP_Create: inicializa la IA del juego
 - APP_Start: controla pause, llama al rungame loop y luego llama a end
 - APP_End: destruye la ventana y todos los demás objetos



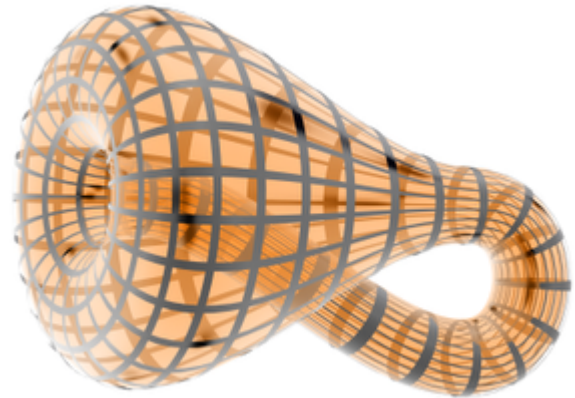
Bucle principal (II)

- RunGameLoop: Da el control a la Inteligencia Artificial del juego, que trabaja a través de los estados de la Inteligencia Artificial
- Estados posibles:
 - CSIG_INITIAL
 - CSIG_FADING_IN
 - CSIG_FADING_OUT2PLAY
 - CSIG_PLAYING
 - CSIG_LOSTLIFE



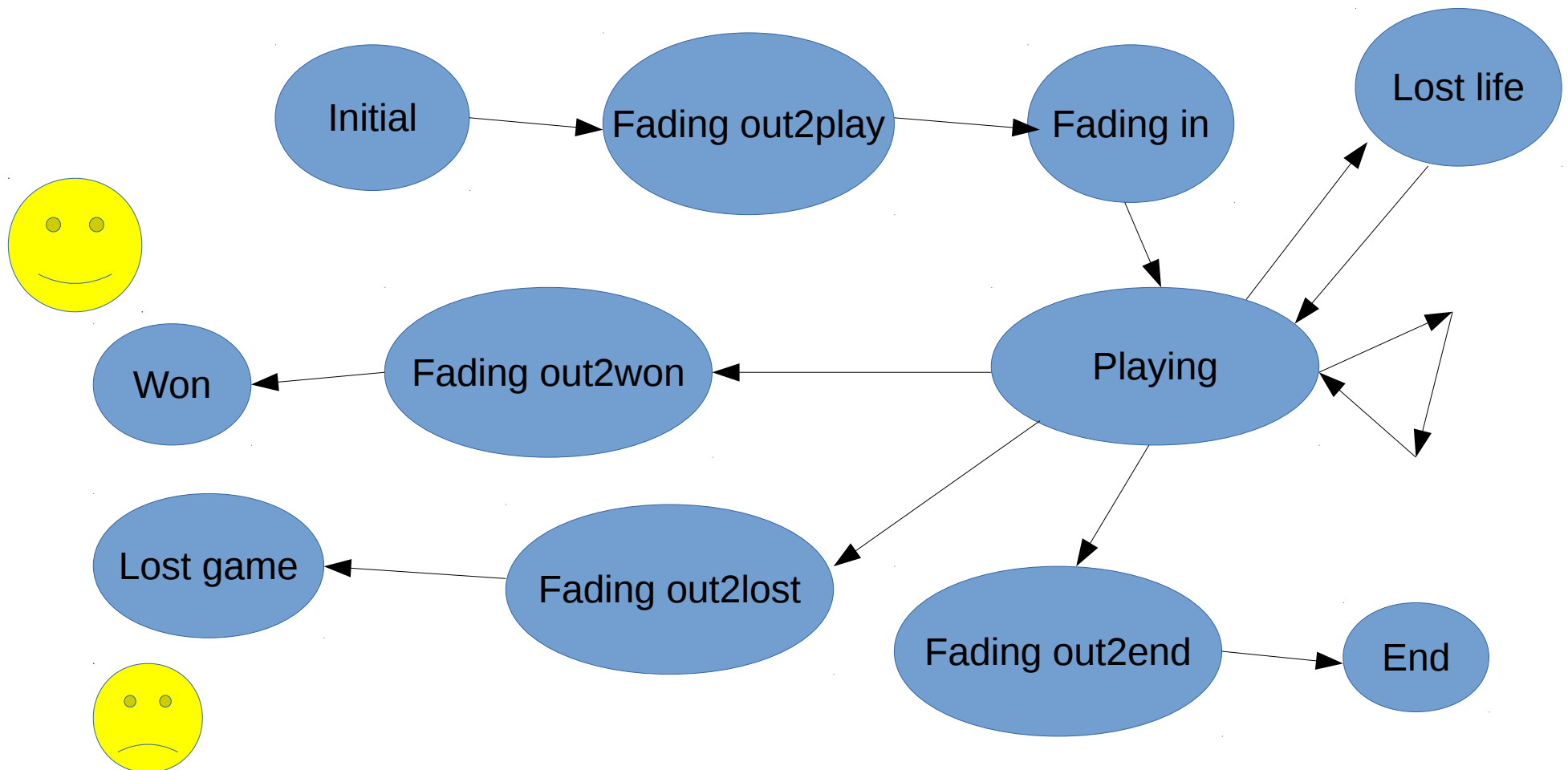
Bucle principal (III)

- Estados posibles:
 - CSIG_WON
 - CSIG_LOSTGAME
 - CSIG_FADING_OUT2WON
 - CSIG_FADING_OUT2END
 - CSIG_FADING_OUT2LOST
 - CSIG_END



Bucle principal (IV)

- Relación entre los estados



Bucle principal (IV)

Estados

- CSIG_INITIAL
 - Configuración inicial del juego
- CSIG_FADING_IN
 - Carga de nivel, carga de lanzamiento de RT-DESK y selección de controles
- CSIG_PLAYING
 - Sincronización de tiempo de RT-DESK y simulación de un paso RT-DESK (Simulate())

Objetos simulados (I)

- Los objetos simulados tienen dos partes principales:
 - Init(): se encarga de la inicialización de sus atributos
 - ReceiveMessage(msg): se encarga de gestionar los mensajes recibidos y actuar en consecuencia

Objetos simulados (II)

Character (I)



- Todos los objetos simulados dependen del objeto character que se encuentra en `character.cpp`
- Se encarga de acciones básicas como son las que listaremos en la siguiente transparencia

Objetos simulados (III)


Character (II)

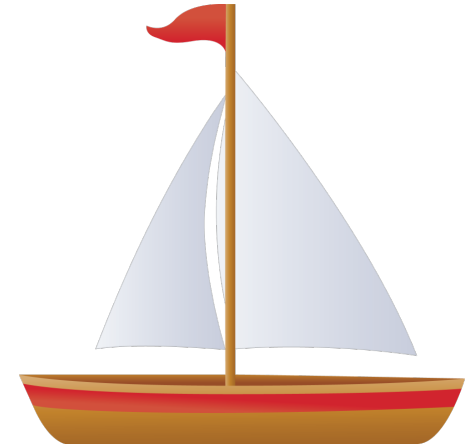
- El objeto character se encarga de:
 - Inicialización
 - Cajas de inclusión
 - Inteligencia artificial
 - RenderMovimiento
 - Colisión



Objetos simulados (IV)

Navy

- In the navy!!! 
- El objeto Navy se encarga de la gestión de las naves enemigas
- El método ReceiveMessage(msg) se encarga de gestionar los mensajes recibidos, actualizar su estado y enviar nuevos mensajes a cada nave, para que actúen en consecuencia



Objetos simulados (V)

Diagrama de clases

- Todas las clases se ubican en el fichero con su nombre y se nombran con una c al principio

