Diseño y Desarrollo de Videjuegos

Justo Miguel Vargas

2013

Universidad Tecnologica Nacional
Facultad Regional Resistencia

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

Diseño y Desarrollo De Videojuegos

El concepto de juego

Dentro del mundo del entretenimiento electrónico, un juego normalmente se suele asociar a la evolución, entendida desde un punto de vista general, de uno o varios personajes principales o entidades que pretenden alcanzar una serie de objetivos en un mundo acotado, los cuales están controlados por el propio usuario. Así, entre estos elementos podemos encontrar desde superhéroes hasta coches de competición pasando por equipos completos de fútbol. El mundo en el que conviven dichos personajes suele estar compuesto, normalmente, por una serie de escenarios virtuales recreados en tres dimensiones y tiene asociado una serie de reglas que determinan la interacción con el mismo.

Game Design Document

El Documento de diseño del juego es la guía escrita con los lineamientos a seguir en todo el diseño, desarrollo, pruebas y evaluación de un videojuego. Es la biblia en la que se consignará TODOS los detalles específicos de tu juego. Sin esto, vamos a ver desordenes y catástrofes en nuestro proyecto.

Si eres programador, te banco cuando leas esto, ya que como desarrolladores siempre queremos ver las cosas funcionando para poder creer en lo que estamos haciendo, pero ojo!, debemos planear antes de ejecutar, es la ley, aunque parezca un poco tedioso, hay que seguir un plan, es realmente importante, nos dará una visión global y más clara de lo que queremos hacer…

Básicamente podemos resumir el documento en módulos o “biblias” de cada uno de los aspectos más importantes de un videojuego, siendo ellos, los siguientes:

- **Biblia Maestra:** Este apartado contiene todos los datos generales del videojuego. El nombre, un resumen de la historia, requerimientos de hardware y software, los miembros del equipo, el cronograma de desarrollo, alcances y objetivos.
- **Biblia del arte:** Contiene todos los desarrollos gráficos del videojuego. StoryBoards, Bocetos de personajes, enemigos, diseño del mundo, items, niveles, concepto artístico y el look and feel.
- **Biblia de la historia:** Dentro de esta va el desarrollo del drama y trama sobre tu videojuego. Lleva una introducción o preludio, la narrativa de la historia, los diálogos de los personajes, las historias y descripciones de cada personaje, enemigos y demás.
- **Biblia del diseño:** Contiene absolutamente todo lo que respecta a jugabilidad. La descripción del gameplay (el como se juega), los controles que se establecieron, mecánicas obstáculos, items, habilidades, desarrollo de niveles.

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

- **Biblia del sonido:** Contiene del desarrollo sonoro del videojuego. Efectos de sonido, pistas de audio, voces y OSTs.
- **Biblia técnica:** Aquí se incluye cada uno de los desarrollos generados para el videojuego. Diagramas de IA, scripts usados, convenciones de lenguajes de programación, algoritmos y si se desarrollara un motor de videojuego se incluiría la estructura de sus engines (sonido, IA, gráficos, etc).

Desde la cátedra te proponemos dos modelos de GDD para que construyas el tuyo a medida con la ayuda del profe. Ver Adjuntos...

Un Vistazo de cómo sería un equipo ideal...

Tu equipo:

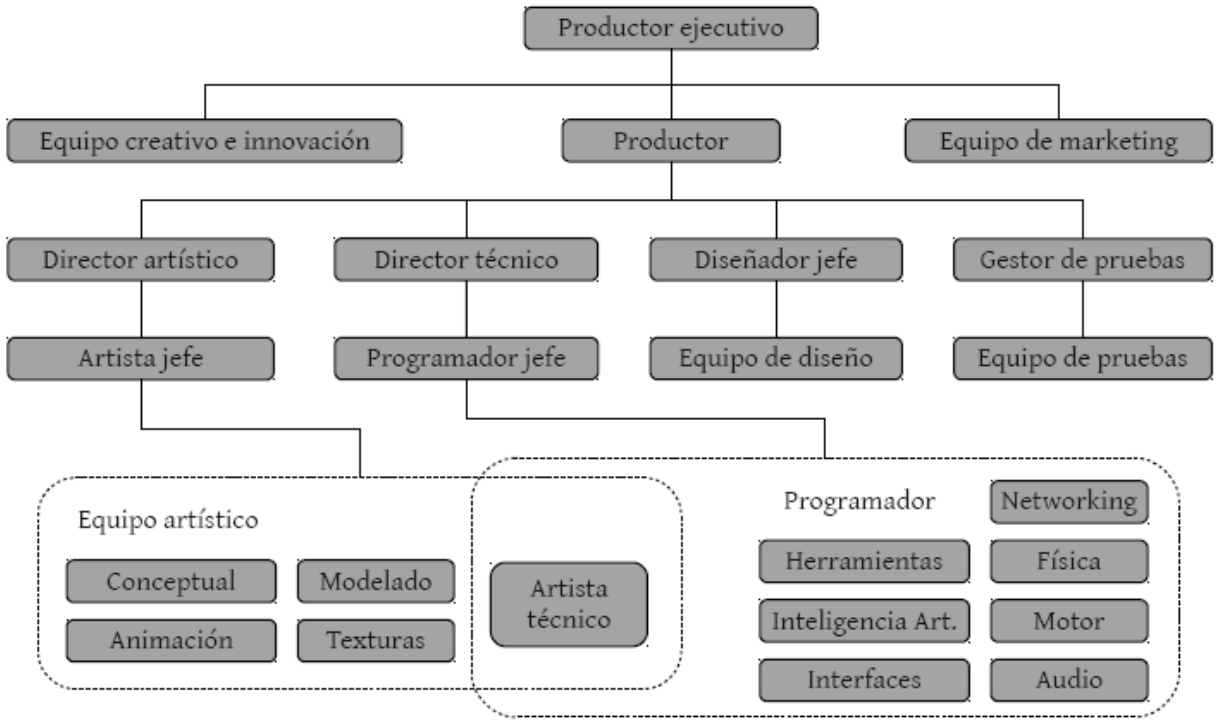


Figura 1.2: Visión conceptual de un equipo de desarrollo de videojuegos, considerando especialmente la parte de programación.

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

1 Equipo de Desarrollo del Diablo III

Arquitectura del motor

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

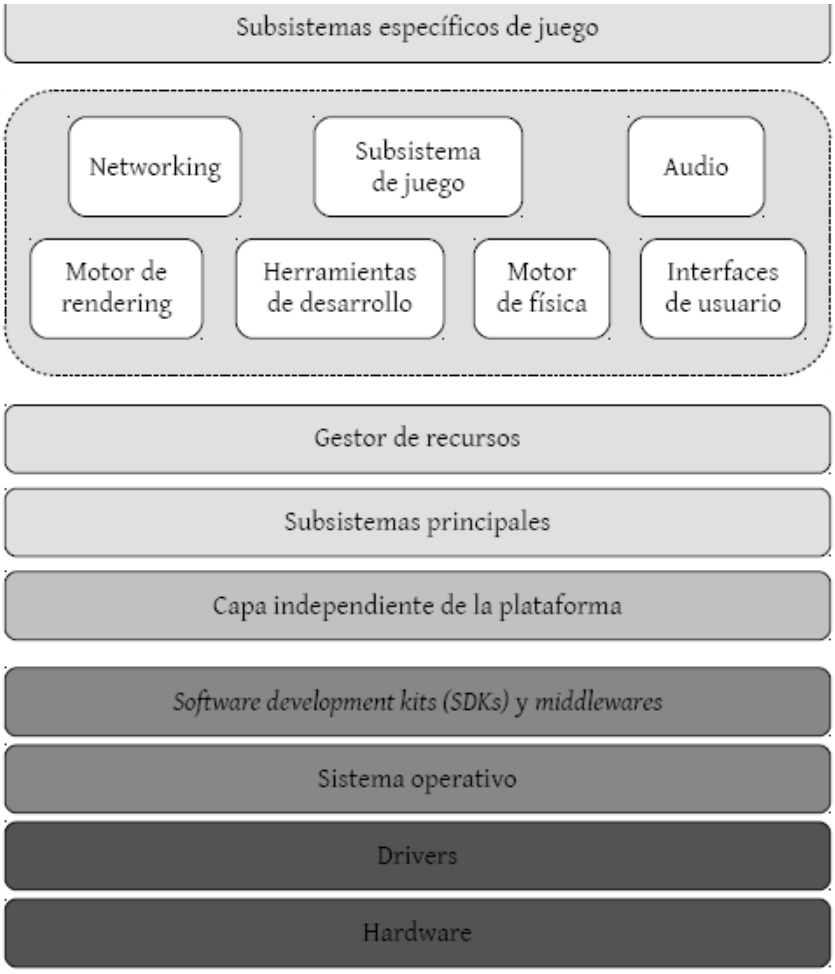


Figura 1.9: Visión conceptual de la arquitectura general de un motor de juegos. Esquema adaptado de la arquitectura propuesta en [5].

Subsistemas Principales:

- **Biblioteca matemática,** responsable de proporcionar al desarrollador diversas utilidades que faciliten el tratamiento de operaciones relativas a vectores, matrices, cuaterniones u operaciones vinculadas a líneas, rayos, esferas y otras figuras geométricas. Las bibliotecas matemáticas son esenciales en el desarrollo de un motor de juegos, ya que éstos tienen una naturaleza inherentemente matemática.
- **Estructuras de datos y algoritmos,** responsable de proporcionar una implementación más personalizada y optimizada de diversas estructuras de datos, como por ejemplo listas enlazadas o árboles binarios, y algoritmos, como por ejemplo búsqueda u ordenación. Este subsistema resulta especialmente importante cuando la memoria de la plataforma o plataformas sobre las que se ejecutará el motor está limitada (como suele ocurrir en consolas de sobremesa).
- **Gestión de memoria,** responsable de garantizar la asignación y liberación de memoria de una manera eficiente.

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

- **Depuración y logging,** responsable de proporcionar herramientas para facilitar la depuración y el volcado de logs para su posterior análisis.

Versionado:

Los juegos realizados para la cátedra se deberán de commitear al siguiente repositorio publico git:

https://github.com/videojuegosfrre/ddv2013.git

Cada grupo deberá de tener credenciales en la pagina de github para poder subir los cambios de código. Para mas información sobre git pueden pegarle un vistazo a:

- http://teach.github.com/articles/course-slides/
- http://githubtraining.s3.amazonaws.com/github-git-training-slides.pdf

Patrones de diseño

¿Que es?

Los patrones de diseño son soluciones generalizadas a los problemas comunes que se producen con frecuencia al crear software utilizando el paradigma de la programación orientada a objetos.

¿Por qué utilizar patrones de diseño?

La respuesta más básica es, que estas soluciones han existido por largo tiempo y muchos expertos los han utilizado, son probablemente mejor que cualquier solución que podrían llegar por su cuenta. E incluso si lo han hecho, lo más probable es que ya exista un patrón de diseño que lo solucione de alguna manera. El tema esta en conocer la mayor cantidad de patrones de diseño, eso nos ayudara a hacer buena arquitectura y plantear buenas decisiones de diseño.

Estructura

Cuando se describe un patrón de diseño se pueden citar más o menos propiedades del mismo: el problema que resuelve, sus ventajas, si proporciona escalabilidad en el diseño o no, etc. Nosotros vamos a seguir las directrices marcadas por los autores del famoso libro de Design Patterns, por lo que para definir un patrón de diseño es necesario describir, como mínimo, cuatro componentes fundamentales:

- Nombre: el nombre del patrón es fundamental. Es deseable tener un nombre corto y autodefinido, de forma que sea fácil de manejar por diseñadores y desarrolladores. Los buenos nombres pueden ser compartidos por todos de forma que se cree un vocabulario común con el que se pueda describir documentación fácilmente, además de construir y detallar soluciones más complejas basadas en patrones. También se debe indicar los alias del patrón.
- Problema y contexto: obviamente, el problema que resuelve un patrón en concreto debe ser descrito detalladamente. Sin embargo, es muy importante que se dé una definición clara del contexto en el que el patrón tiene sentido aplicarlo. *El contexto se puede ver como un listado de precondiciones que deben ser cumplidas para poder aplicar el patrón.*
- Solución: la solución que proporciona un patrón se describe genéricamente y nunca ligada a ninguna implementación. Normalmente, se utiliza los conceptos y nomenclatura de la programación orientada objetos. Por ello, la solución normalmente describe las clases y las relaciones entre objetos, así como la responsabilidad de cada entidad y cómo colaboran entre ellas para llegar a la solución. Gracias a la adopción de esta nomenclatura, la implementación de los patrones en los

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

lenguajes orientados a objetos como C++ es más directa dada su especificación abstracta.

- Ventajas y desventajas: la aplicación de un patrón de diseño no es una decisión que debe tomarse sin tener en cuenta los beneficios que aporta y sus posibles inconvenientes. Junto con los anteriores apartados, se deben especificar las ventajas y desventajas que supone la aplicación del patrón en diferentes términos: complejidad, tiempo de ejecución, acoplamiento, cohesión, extensibilidad, portabilidad, etc. Si estos términos están documentados, será más sencillo tomar una decisión.

Los mas comunes...

- **Singleton** - Se utiliza cuando necesitamos que una única instancia de un objeto es pemritida durante todo el juego. Por ejemplo, en cocos2d , existen clases singleton, como CCDirector, CCSpriteFrameCache , etc Al parecer se trata de una panacea de uso frecuente en la programación de juegos, aunque puede enmascarar una arquitectura mala. Probablemente se deberia evitar el uso de este modelo de diseño si se puede , porque es probable que exista una mejor manera de diseñar la arquitectura de nuestros juegos.
- **Factory** - Se crea un objeto cuyo propósito es crear otros objetos. Por ejemplo, se puede tener una clase “fábricadora” llamada "GameObjectFactory" con métodos estáticos (posiblemente con parámetros) para crear otros objetos de juego como un "jugador", "enemigo", "Gun" , o "Bullet", o bien varias subfabricas. Se utiliza cuando las clases a fabricar tienen construcciones complejas que hacen que la obtención de una instancia de esa clase específica sea difícil o al menos no tan simple. La fábrica puede ademas añadirla a un pool de objetos, o bien al motor de fisica del mismo y devolver una referencia al objeto creado. Este patrón ayuda a evitar el problema de la creación de instancias de objetos complejos, manteniendo estas configuraciones complejas en un solo lugar, en lugar de tenerlos esparcidos alrededor del código.
- **Observador** - El objeto en cuestión mantiene una lista de otros objetos que estén interesados en su estado y notifica estos objetos de escucha de un cambio en su estado. Explicado mas abajo.
- **Estado** - Se tiene una clase abstracta (aplicación vacía) que tiene subclases para definir el estado actual. Un ejemplo de esto podría ser un shooter en primera persona que tiene una clase de "jugador" que tiene varios estados posibles como "PlayerInCombat", "PlayerOutOfCombat" , y "PlayerInMenu". Este patrón nos ayuda a evitar los métodos monolíticos que realizan diferentes acciones en función del estado del objeto que tiene un montón de ifs anidados o switches.Explicado en detalle mas abajo.

Se adjunta como material de lectura el libro ActionScript.3.0.Design.Patterns donde podran encontrar implementaciones de los patrones de diseños mas comunes a la hora de realizar un juego. Si bien la implementaciones hechas

estan sobre AS3, se pueden trasladar tranquilamente a cualquier lenguaje. Se recomienda ver los patrones **Decorator** y el **Strategy**.

Main Loop & Sprites

Marco Teórico:

Desde una perspectiva más formal, la mayoría de videojuegos suponen un ejemplo representativo de lo que se define como aplicaciones gráficas o renderizado en tiempo real [1], las cuales se definen a su vez como la rama más interactiva de la Informática Gráfica. Desde un punto de vista abstracto, una aplicación gráfica en tiempo real se basa en un bucle donde en cada iteración se realizan los siguientes pasos:

- El usuario visualiza una imagen renderizada por la aplicación en la pantalla o dispositivo de visualización.
- El usuario actúa en función de lo que haya visualizado, interactuando directamente con la aplicación, por ejemplo mediante un teclado.
- En función de la acción realizada por el usuario, la aplicación gráfica genera una salida u otra, es decir, existe una retroalimentación que afecta a la propia aplicación.

En este contexto, el ***frame rate*** se define como el número de imágenes por segundo, comúnmente fps, que la aplicación gráfica es capaz de generar. A mayor frame rate, mayor sensación de realismo en el videojuego. Actualmente, una tasa de 30 fps se considera más que aceptable para la mayoría de juegos. No obstante, algunos juegos ofrecen tasas que doblan dicha medida.

Caída de frames

Si el núcleo de ejecución de un juego no es capaz de mantener los fps a un nivel constante, el juego sufrirá una caída de frames en un momento determinado. Este hecho se denomina comúnmente como ralentización. En el caso de los videojuegos, este ciclo de visualización, actuación y renderizado ha de ejecutarse con una frecuencia lo suficientemente elevada como para que el usuario se sienta inmerso en el videojuego, y no lo perciba simplemente como una sucesión de imágenes estáticas.

Esquema general de un bucle de renderizado.

```
1 while (true) {
2 // Actualizar la cámara,
3 // normalmente de acuerdo a un camino prefijado.
4 update_camera ();
5
6 // Actualizar la posición, orientación y
7 // resto de estado de las entidades del juego.
8 update_scene_entities ();
9
10 // Renderizar un frame en el buffer trasero.
11 render_scene ();
```

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

```
12
13 // Intercambiar el contenido del buffer trasero
14 // con el que se utilizará para actualizar el
15 // dispositivo de visualización.
16 swap_buffers ();
17 }
```

Rectangle invalidation

La técnica basada en redibujar únicamente aquellas partes de la pantalla cuyo contenido cambia realmente se suele denominar rectangle invalidation.

Keep it simple, Stupid!

La filosofía KISS (Keep it simple, Stupid!) se adapta perfectamente al planteamiento del bucle de juego, en el que idealmente se implementa un enfoque sencillo, flexible y escalable para gestionar los distintos estados de un juego.

Ejemplo Esquema general del bucle de juego.

```
1 // Pseudocódigo de un juego tipo "Pong".
2 int main (int argc, char* argv[]) {
3 init_game(); // Inicialización del juego.
4
5 // Bucle del juego.
6     while (1) {
7           capture_events(); // Capturar eventos externos.
8
9           if (exitKeyPressed()) // Salida.
10                break;
11
12          move_paddles(); // Actualizar palas.
13          move_ball(); // Actualizar bola.
14          collision_detection(); // Tratamiento de colisiones.
15
16          // ¿Anotó algún jugador?
17          if (ballReachedBorder(LEFT_PLAYER)) {
18                score(RIGHT_PLAYER);
19                reset_ball();
20          }
21          if (ballReachedBorder(RIGHT_PLAYER)) {
22                score(LEFT_PLAYER);
23                reset_ball();
24          }
25
26          render(); // Renderizado.
27    }
28 }
```

De los juegos descritos anteriormente identificar:

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

1. Identifique el Main loop de cada uno. Sabiendo que tenemos diferentes formas de implementar esto (Por eventos o por tiempo) en los juegos, responda a su criterio:
   - ¿Como lo implementaría en cada uno de los escenarios plateados?
   - ¿Que elementos debería controlar en el mismo? En caso de existir dos formas de hacerlo, explique cual seria la mejor y por que.
   - Proponga un diagrama de flujo sencillo para el main loop de alguno de los juegos (Omita controles gráficos o de sonido extras).

2. Realice un informe conteniendo lo siguiente:
   - Seleccione 4 plataformas distintas de programación videojuegos y explique como tratan al Main Loop cada una empleando para ello ejemplos realizados con juegos existentes.
   - Explique la diferencia entre un time-step fijo y otro variable. ¿Que juegos son mas fáciles de desarrollar con cada uno? Encuentre 4 juegos que emplean cada técnica. Explique como simular un step fijo a través de un sistemas que ofrece time-step variables.
   - Identifique al menos 3 sprites en cada juego. Seleccione un plataforma de programación y explique como maneja la misma el manejo de Sprites, si esta la soporta sino explique como lo haría desde el main loop. Reproduzca el sprite en el formato de imagen que considere adecuado según la plataforma seleccionada. Fundamente la elección si esta posee varios permitidos.

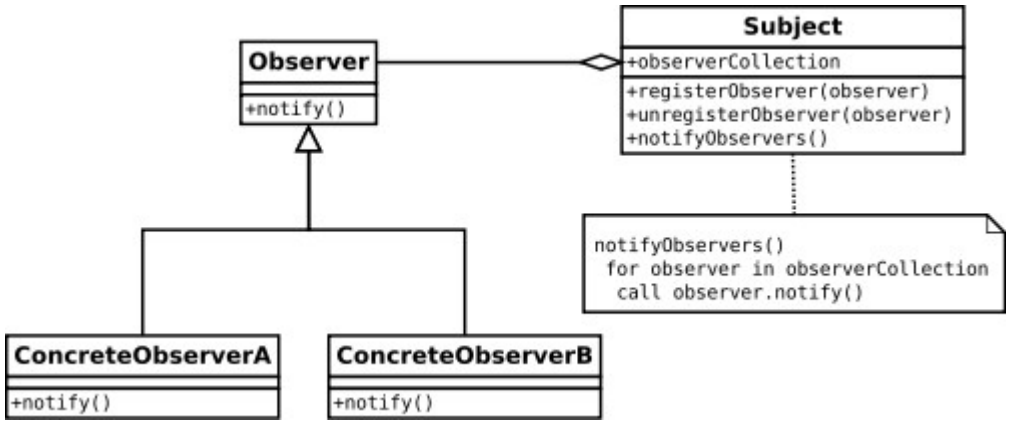
French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

State Machines & Programación Orientada a Eventos

Tratamiento de Eventos:

En el ámbito de los juegos, un evento representa un **cambio** en el estado del propio juego o en el entorno. Un ejemplo muy común está representado por el jugador cuando pulsa un botón del joystick, pero también se pueden identificar eventos a nivel interno, como por ejemplo la reaparición o respawn de un NPC en el juego.

Gran parte de los motores de juegos incluyen un subsistema específico para el tratamiento de eventos, permitiendo al resto de componentes del motor o incluso a entidades específicas registrarse como partes interesadas en un determinado tipo de eventos. Este planteamiento está muy estrechamente relacionado con el patrón Observer[1].



El tratamiento de eventos es un aspecto transversal a otras arquitecturas diseñadas para tratar el bucle de juego, por lo que es bastante común integrarlo dentro de otros esquemas más generales, como por ejemplo el que se discute a continuación y que está basado en la gestión de distintos estados dentro del juego.

Esquema basado en estados

Desde un punto de vista general, los juegos se pueden dividir en una serie de etapas o estados que se caracterizan no sólo por su funcionamiento sino también por la interacción con el usuario o jugador.

Típicamente, en la mayor parte de los juegos es posible diferenciar los siguientes estados:

- **Introducción o presentación**, en la que se muestra al usuario aspectos generales del juego, como por ejemplo la temática del mismo o incluso cómo jugar.
- **Menú principal**, en la que el usuario ya puede elegir entre los distintos modos de juegos y que, normalmente, consiste en una serie de entradas textuales identificando las opciones posibles.

[1] Ver Patron de Diseño

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

- **Juego**, donde ya es posible interactuar con la propia aplicación e ir completando los objetivos marcados.
- **Finalización o game over**, donde se puede mostrar información sobre la partida previamente jugada.

Por otra parte, existe una **relación** entre cada uno de estos estados que se manifiesta en forma de **transiciones** entre los mismos. Por ejemplo, desde el estado de **introducción** sólo será posible acceder al estado de **menú principal**, pero no será posible acceder al resto de estados. En otras palabras, existirá una **transición** que va desde introducción a menú principal.

Otro ejemplo podría ser la transición existente entre finalización y menú principal.

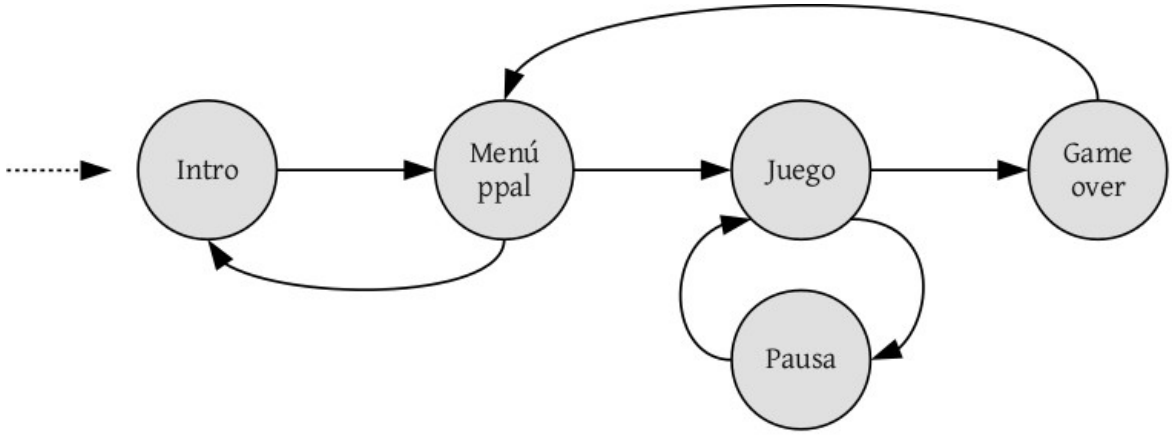


Figura 6.5: Visión general de una máquina de estados finita que representa los estados más comunes en cualquier juego.

Implementaciones de Maquinas de Estados

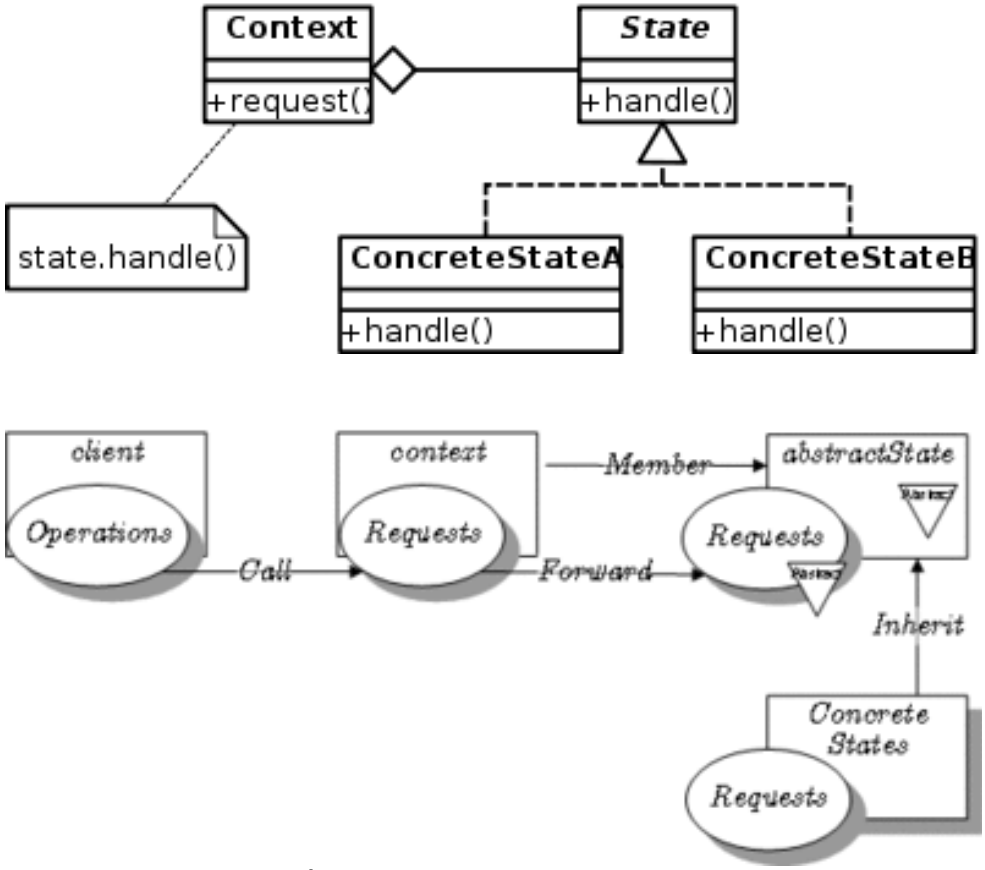
1. Números Mágicos

```
1 /Implementacion de estados de un Juego mediante Números Mágicos
2
3     final int INIT_GAME = 1;
4     final int PLAYING = 2;
5.    final int END_GAME = 3;
6
7     private int state;
8
9
10 int main (int argc, char* argv[]) {
11          switch(state){
12
13
14                INIT_GAME:
15                      //some cool code
16                break;
17
18                PLAYING:
19                      //some cool code
20                break;
```

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

```
21
22                  END_GAME:
23                        //some cool code
24                  break;
25
26            }
27      }
28 }
```

2. Implementaciones Propias o Por Patrones



Una implementación minimalista:

```
interface Statelike {

   /**
    * Writer method for the state name.
    * @param STATE_CONTEXT
    * @param NAME
    */
    void writeName(final StateContext STATE_CONTEXT, final String
NAME);

}
```

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

```
class StateA implements Statelike {
   /* (non-Javadoc)
            *  @see  state.Statelike#writeName(state.StateContext,
java.lang.String)
    */
   @Override
    public void writeName(final StateContext STATE_CONTEXT, final
String NAME) {
      System.out.println(NAME.toLowerCase());
      STATE_CONTEXT.setState(new StateB());
   }

}
```

```
class StateB implements Statelike {
   /** State counter */
   private int count = 0;

   /* (non-Javadoc)
            *  @see  state.Statelike#writeName(state.StateContext,
java.lang.String)
    */
   @Override
    public void writeName(final StateContext STATE_CONTEXT, final
String NAME) {
      System.out.println(NAME.toUpperCase());
      // Change state after StateB's writeName() gets invoked twice
      if(++count > 1) {
         STATE_CONTEXT.setState(new StateA());
      }
   }

}
```

```
public class StateContext {
   private Statelike myState;
      /**
       * Standard constructor
       */
   public StateContext() {
      setState(new StateA());
   }

      /**
       * Setter method for the state.
          * Normally only called by classes implementing the State
interface.
       * @param NEW_STATE
```

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

```
        */
    public void setState(final Statelike NEW_STATE) {
        myState = NEW_STATE;
    }

        /**
         * Writer method
         * @param NAME
         */
    public void writeName(final String NAME) {
        myState.writeName(this, NAME);
    }
}
```

Practica:

- De acuerdo al juego asignado:
- Realice una lista de eventos posibles en el juego, y de los despachadores de los mismos.
- Identifique posibles estados generales del juego, y sub-estados dentro de los mismos.
- Esquematice un modelo de máquina de estados con lo generado anteriormente. Explique con detalle cuales serian los eventos para pasar de un estado a otro y la condición de fin.

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

Gestión eficiente de la memoria

Dados los siguientes juegos y sus descripciones:

- ✓ **ZUMA:** El juego deriva en la época de la cultura azteca. El juego consiste en que el jugador debe formar grupos de 3 o más bolas del mismo color alrededor de una pista, antes de que la cadena de bolas llegue a la Calavera, el punto donde desaparecen.
- ✓ **Bejeweled**: El objetivo del juego es intercambiar una joya con una adyacente para formar una cadena de tres o más joyas del mismo color. Cuando una cadena se crea, las joyas desaparecen, provocando que aleatoriamente caigan joyas desde la parte superior que tendrá para que las joyas se vayan emparejando. A veces, algunas joyas se combinan automáticamente al caer, a lo que se le denomina *cascada*.
- ✓ **Bouncing Balls:** el objetivo del juego es terminar con todas las bolitas de colores disparando con otra del mismo color antes que la pila de estas toca el piso. Para desaprecer debes juntar mas de dos bolas del mismo color.
- ✓ **Asteroids:** es un popular videojuego de arcade basado en vectores lanzado en 1979 por Atari. El objetivo del juego es disparar y destruir asteroides evitando chocar contra los fragmentos de estos. Fue uno de los juegos más populares de la Época Dorada de los videojuegos arcade.

En Grupos de 4 a 6 integrantes:

a) Proponga al menos dos estructuras de datos que considere conveniente para la lógica principal de los mismos y compárelas a su criterio con respecto al uso de memoria de cada una de ellas.

b) ¿Que problemática de gestión de memoria esta por detrás de estos juegos?

c) ¿Qué modificaciones haría para mejorar esta situación?

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

Discutir grupal sobre las diferentes metodologías/técnicas de gestión de memoria:

- Garbage collector
- Pool de objetos
- Asignación dinámica

Para cada uno de los juegos descritos seleccionar una técnica o combinación de estas para el desarrollo del juego y defiéndala en clase.

Pool de Objetos[2]

En resumen un pool de objetos nos permite recuperar una instancia de un tipo particular de objeto mediante la extracción dentro y fuera de un “pileta de recursos. Aunque el manejo de la situación en la que un grupo está vacío no lo está.

Hay tres enfoques voy a hablar de la manipulación de una piscina vacía:

1) Pool sin límites

La piscina tiene un tamaño ilimitado y objetos se agregan según sea necesario para el grupo de recursos a la salida. Esto asegura que un objeto siempre será devuelto a la salida.

2) Pool con topes

El pool tiene un tamaño constante y el grupo de recursos se rellena durante la construcción del pool. Esto le permite tener toda la sobrecarga de la memoria en la construcción del pool y no en la confirmación.

3) Pool con colas

El pool tiene un tamaño mínimo y el fondo de recursos se llena durante la construcción como una opción tiene la ventaja de un tamaño máximo - permitiendo al pool para crecer.

[2] Consultar Material de la Cátedra para ver un Pool Implementado

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

Exteriorizacion de Variables

- Enumere cuales son las ventajas/desventajas del HARDCODING. ¿De que forma se puede resolver?
- Seleccionar uno de los juegos propuestos y definir:
  - Variables que se exteriorizaran
  - Variables que deberían permanecer como constantes en el juego.
- Investigue los diferentes formatos existentes para la lectura de variables externas. Seleccione una plataforma y explique como esta trata el problema.
-

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

Marco Teorico:

Tipos de Narrativa:

3. ***Narrativa Lineal***
   - Esquema: Exposicion → Conflicto → Resolucion
   - Ejemplos: Super Marios Bross (Nintendo), Zuma, Max Payne
4. ***Narrativa Abierta o Mundo Abierto***
   - Esquema: No tiene flujo asociado...
   - Ejemplos: Worl of Warcarft, Mario 64
5. ***Narrativa Ramificada***
   - Esquema: Exposicion → Conflicto1 → Resolucion1
     → Conflicto2 → Resolucion2
     → Conflicto3 → Resolucion3
     → ..... → .....
     → ConflictoN → ResolucionN
   - Ejemplos: *Heavy Rain*, Star Wars
6. ***Narrativa Paralela***
   - Esquema:

Exposicion → Conflicto1 → Conflicto Secundario → Conflicto1 → Resolucion Final

→ Conflicto2 →	→ Conflicto1 →
→ Conflicto3 →	→ Conflicto1 →
→ ..... →	→ ..... →
→ ConflictoN →	→ Conflicto1 →

   - Ejemplos: Syberia, The Sims

Practica:

Dados los siguientes juegos, definir que tipo de narrativa tienen.

- Castlevania
- Grand Theft Auto
- Metal Gear
- Megaman
- Zelda 64
- Batman Arkham City
- ProEvolution Soccer
- Age of Empires

Manos a la Obra! Es momento de hacer volar la imaginacion...

- Desarrolle una narrativa/ historia de juego lo mas original posible (Describala como una historia con cuerpo definido y formal. Trate de

French 414
Tel 03722-432928 - Fax 03722-432683
(3500) Resistencia Chaco

Hacerla Interesante al publico y que sea atrapante, no se limite con la imaginacion).

- Que tipo de Narrativa usaste? Por que?
- ¿A que genero cree que pertenece esta narrativa planteada?

Luego de definir lo anterior, y tomandolo como base dessarolle lo siguiente:

- Plantear personajes, jugabilidad, plataformas.
- Esquematize en simples scketches las posibles pantallas del juego.
- Desarolle un diagrama de navegacion entre estas pantallas.
- Indetifique en su juego:
  1. Competencia.
  2. Condiciones de victoria.
  3. Configuraciones.
  4. Modelo de interacción.
- ¿Que experiencias/servicios externas al juego se podrian ofrecer al usuario?
- Describa el equipo de trabajo que considere adecuado para llevar a cabo el proyecto de implementiacion del juego.
- Estime tiempo y costo de desarrollo.