DISEÑO Y ARQUITECTURA DE VIDEOJUEGOS

Andrés Pastorini - Alvaro Martínez

TDIVJ2D – Tecnólogo Informático

TEMARIO

1. **Diseño conceptual de videojuegos**
2. Diseño del sistema

1. Diseño conceptual de video juegos

No hay recetas para el diseño de videojuegos.

1.1 Juegos

Juegos

- Dos conceptos que van de la mano:
  Juegos ~ Diversión

- **Definición de diversión:**
  Acción y efecto de divertir. Entretener, recrear.

- La diversión es una experiencia emocional.

1.1 Juegos

Los mamíferos jóvenes aprenden mediante juegos las **habilidades** básicas para la supervivencia.

1.1 Juegos

- La diversión puede verse como la practica o **aprendizaje** de nuevas **habilidades** en un entorno relativamente seguro.

- Cuando las personas dejan de aprender de un juego dejan de jugarlo.

1.1 Juegos

Categorías de habilidades

- **Físicas**
  (deportes, fuerza, coordinación, exploración y descubrimiento, uso de herramientas, etc.).
- **Sociales**
  (relacionamiento, liderazgo, asociación, transferencia de conocimiento, coqueteo, etc.).
- **Mentales**
  (razonamiento abstracto, reconocimiento de patrones, juegos de palabras, discernimiento, etc.).

- En los juegos se combinan las distintas categorías.

1.2 Diseño de Videojuegos

Para diseñar videojuegos es importante tener en cuenta

- ¿Que hace a un videojuego divertido?
- ¿Porque la gente juega videojuegos?

Las respuestas no son las mismas para todas las personas.

1.2 Diseño de Videojuegos

Tipos de vidieojugadores

- **Casuales**
  Se detienen cuando el juego deja de ser divertido.
  Los retos deben ser razonables.

1.2 Diseño de Videojuegos

Tipos de vidieojugadores

- **Gamers/Core Gamers**
  Casi siempre finalizan los juegos que juegan.
  Buscan juegos difíciles, toleran la frustración.

1.2 Diseño de Videojuegos

Tipos de vidieojugadores

- **Pro-Gamers**
  Juegan por dinero.

1.2 Diseño de Videojuegos

Duración de juego

- Se puede definir una unidad significativa de juego como la cantidad mínima de tiempo que se debe jugar para divertirse.

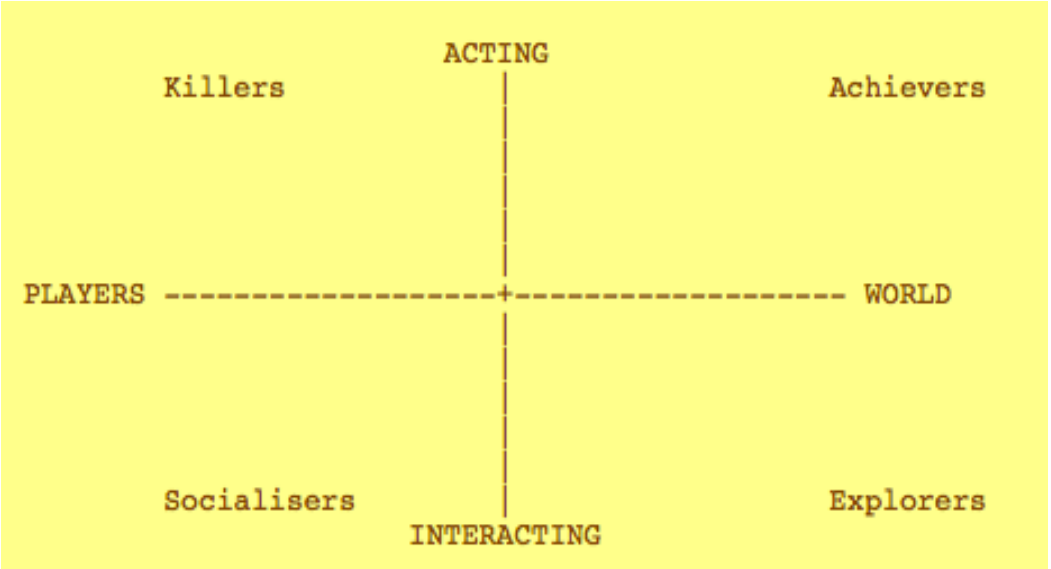
  Consolas: 30 minutos
  Celulares: 1 minuto

- Los jugadores casuales se caracterizan por jugar generalmente cortos períodos de tiempo.
  Esto influye de forma significativa en el diseño del juego.

1.2 Diseño de Videojuegos

Los cuatro tipos de Bartle

- Teoría que clasifica los jugadores online en categorías según los intereses de los mismos.



1.2 Diseño de Videojuegos

- **Archiver:** superar los retos, recoger premios.
- **Explorer:** interesados en descubrir, entender el mundo de juego.
- **Socializer:** Interactuar con el resto, interpretar un rol.
- **Killer:** desarrollar sus habilidades, vencer a los demás.
- Los juegos exitosos apelan a varios grupos.
- World of Warcraft fue diseñado para los cuatro.

1.2 Diseño de Videojuegos

1.2 Diseño de Videojuegos

Definiciones de juegos (otra vez)

Un juego es una forma de entretenimiento interactivo en el que los **jugadores** deben superar los **retos**, por medio de acciones que se rigen por **reglas**, con el fin de cumplir con una **condición de victoria**.

Un juego es un sistema en el que los **jugadores** se enfrentan en un **conflicto** artificial, definido por **reglas**, que se traduce en un resultado **cuantificable**.

1.3 Proceso de Diseño

- **Imaginar el juego:** visión artística global.
- **Definir la forma en que funciona:** Jugadores, Retos, Reglas, Metas.
- **Describir los elementos del juego:** Especificación del gameplay [*].
- **Transmitir esa información:** Documentos de diseño, guía para el equipo de desarrollo.

1.3 Proceso de Diseño

Definición de Gameplay

Incluye todas las experiencias de los jugadores durante la interacción con los sistemas de juego. El gameplay es lo que el jugador hace.

1.3 Proceso de Diseño

Decisiones sobre componentes de diseño

- Jugadores:
  ¿Cuántos jugadores al mismo tiempo?
  ¿Quién o qué es el jugador en el universo del juego?

- Historia (Storytelling)
  ¿Cual es la historia?
  ¿El jugador puede cambiarla?
  ¿Cuál es la conexión entre la historia y el gameplay?

1.3 Proceso de Diseño

Decisiones sobre componentes de diseño

- Retos
  ¿Cuáles son los obstáculos que debe superar el jugador?
  ¿Hay más de una forma de superarlos?

- Reglas
  ¿Cómo interactúa el jugador con el universo?
  ¿Cómo aprende las reglas el jugador?
  ¿Cuáles son los limites del juego?

Retos y Reglas definen los fundamentos del gameplay

1.3 Proceso de Diseño

Decisiones sobre componentes de diseño

- Metas
  ¿Cuáles son las condiciones de triunfo?
  ¿Definidas por el juego o por el jugador?

- Configuración
  ¿Cómo ve el juego el jugador ?
  ¿Cuál es el método de interacción del jugador con el juego?

- Otros...

1.3 Proceso de Diseño

Enfoques para diseñar juegos

- Centrado en el género
- Centrado en el gameplay
- Centrado en la experiencia del usuario

1.3 Proceso de Diseño

Centrado en el género

Se trata de usar un género para definir el juego. Lo más importante a definir es la historia y aspectos de configuración, el resto lo provee el género.

1.3 Proceso de Diseño

Centrado en el género

Ventajas:

- Los géneros dan un marco de gameplay bien conocido.
- Con nuevos escenarios y algunos detalles se puede hacer ver al juego como original.
- Buenos para diseñar pensando en una audiencia en particular.

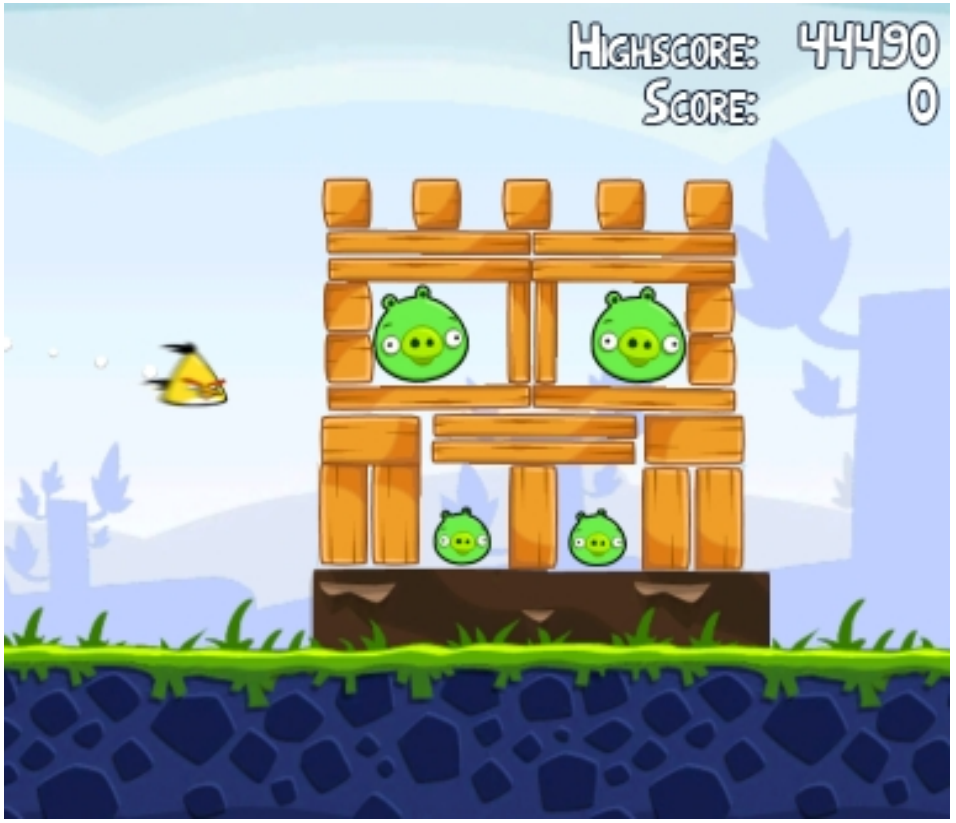
Desventajas:

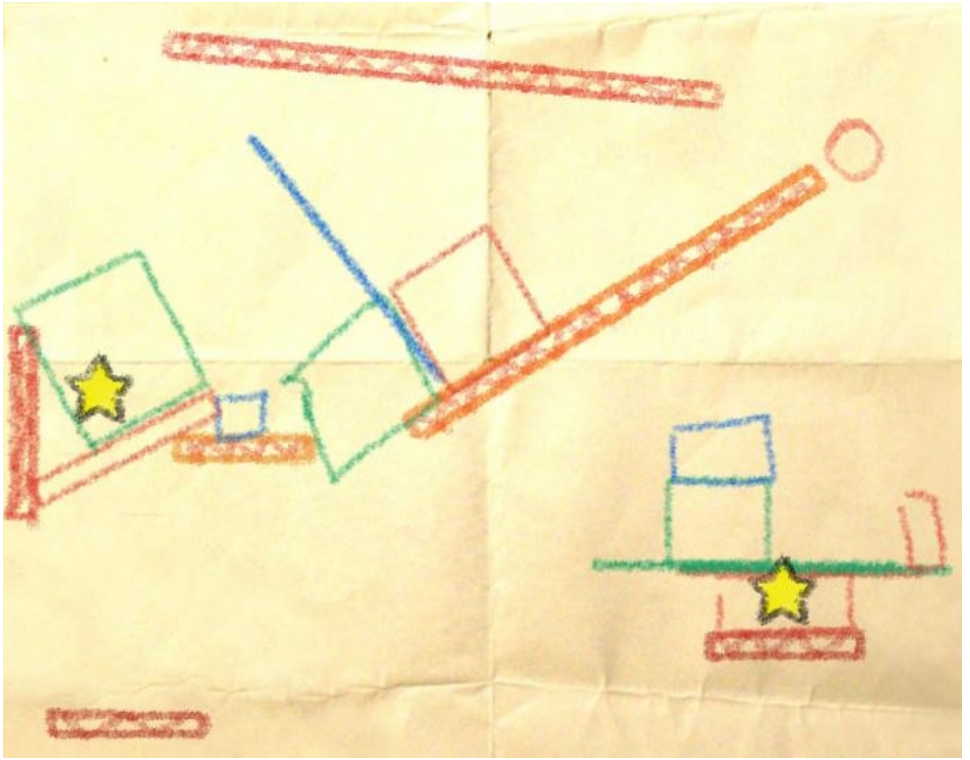
- Fácil de quedar atrapados en las convenciones del genero.

1.3 Proceso de Diseño

Centrado en el gameplay

Se trata de diseñar el juego pensando en los nuevos retos y la mecánica del juego.





1.3 Proceso de Diseño

Centrado en el gameplay

Ventajas:

- Es la mecánica del juego, no los escenarios, los que hacen al juego original.
- Usualmente sencillos y elegantes.
- Muy exitosos en el campo de los juegos casuales.

Desventajas:

- Pueden derivar en escenarios ya vistos o demasiado abstractos
- Difíciles de vender a los editores.

1.3 Proceso de Diseño

Centrado en la experiencia del usuario

Diseñar pensando en la experiencia del usuario. Lo más importante es la interacción.

1.3 Proceso de Diseño

Centrado en la experiencia del usuario

Ventajas:

- Se logran juegos muy coherentes.
- Da una excelente visión general del juego.

Desventajas:

- Difícil de traducir en una mecánica concreta de juego.
- A menudo llevan mayor tiempo de desarrollo.

TEMARIO

1. Diseño conceptual de videojuegos
2. **Diseño del sistema**

2 Diseño y Arquitectura de Videojuegos

Se pasa del diseño conceptual al diseño de componentes.

2.1 Diseño de Alto Nivel.
2.2 Diseño de Componentes.
2.3 Main Loop.
2.4 Manejo del Tiempo.
2.5 Elementos de Diseño.
2.6 Ejemplo de Arquitectura.
2.7 Data Driving.

2.1 Diseño de Alto Nivel

Preguntas

¿Que tiene o usa el juego?

¿Que hace, cuando y con que frecuencia?

2.1 Diseño de Alto Nivel

Cosas que el juego tiene

- **Front-end:** Interacción con el usuario, menús, métodos de entrada.
- **Gráficos:** animación, fondo.
- **Sonidos:** efectos especiales, música.
- **Objetos:**
  En el juego: elementos, plataformas, personajes
  HUD: puntajes, vidas, tiempo.
- **Contenedores de objetos:**
  Niveles, áreas, mapas.
  Eventos de entrada.

2.1 Diseño de Alto Nivel

Cosas que el juego hace

- **Una vez por nivel/juego:**
  Carga los elementos gráficos y de sonido.
  Construye los objetos.
  Puebla los contenedores de objetos.
- **Todo el tiempo:**
  Considerar eventos de entrada.
  Actualizar objetos.
  Dibujar los gráficos.
  Reproducir los sonidos.

2.2 Componentes

Orientación a objetos

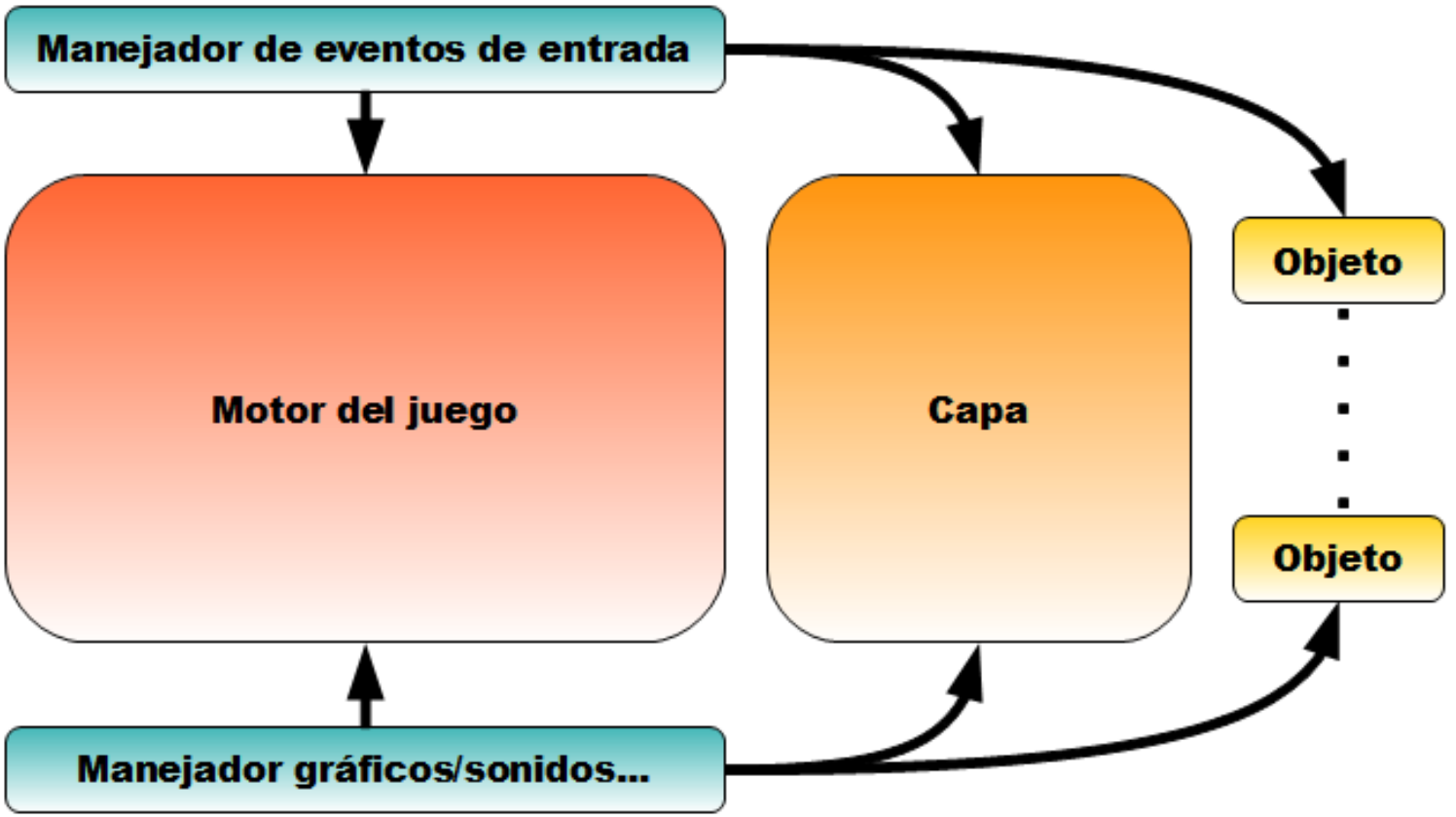
- Abstracción.
- Encapsulamiento.
- Modularidad.
- Polimorfismo.
- Herencia.

2.2 Componentes

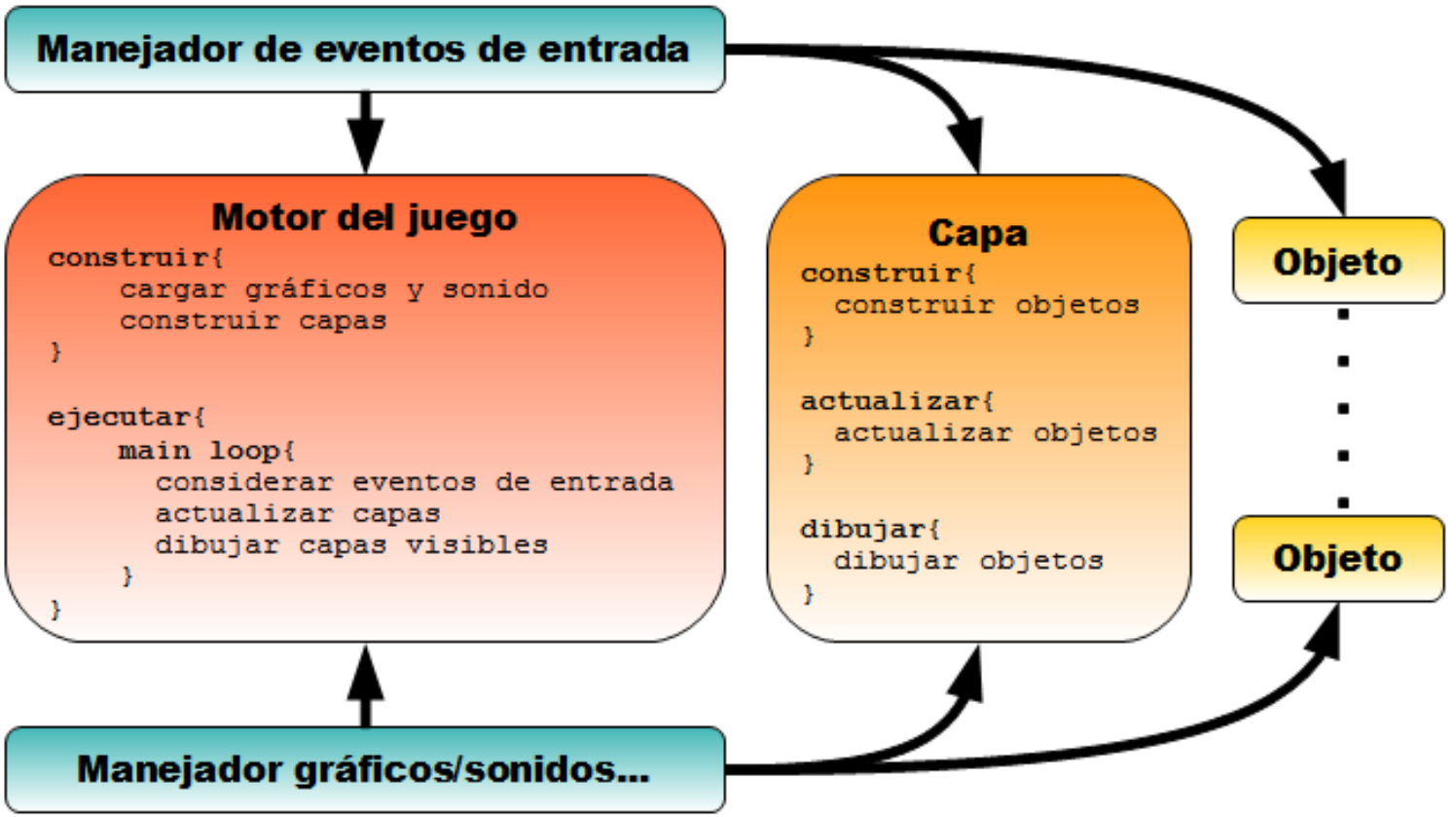
Orientación a objetos

- Podemos diseñar un juego con un enfoque de capas.
- El juego está compuesto por **capas**, donde cada capa tiene **objetos**.
- Otros componentes del juego serán:
  Manejadores (sonido, gráficos, eventos de entrada)
  Motor del juego (incluye el main loop)

2.2 Componentes



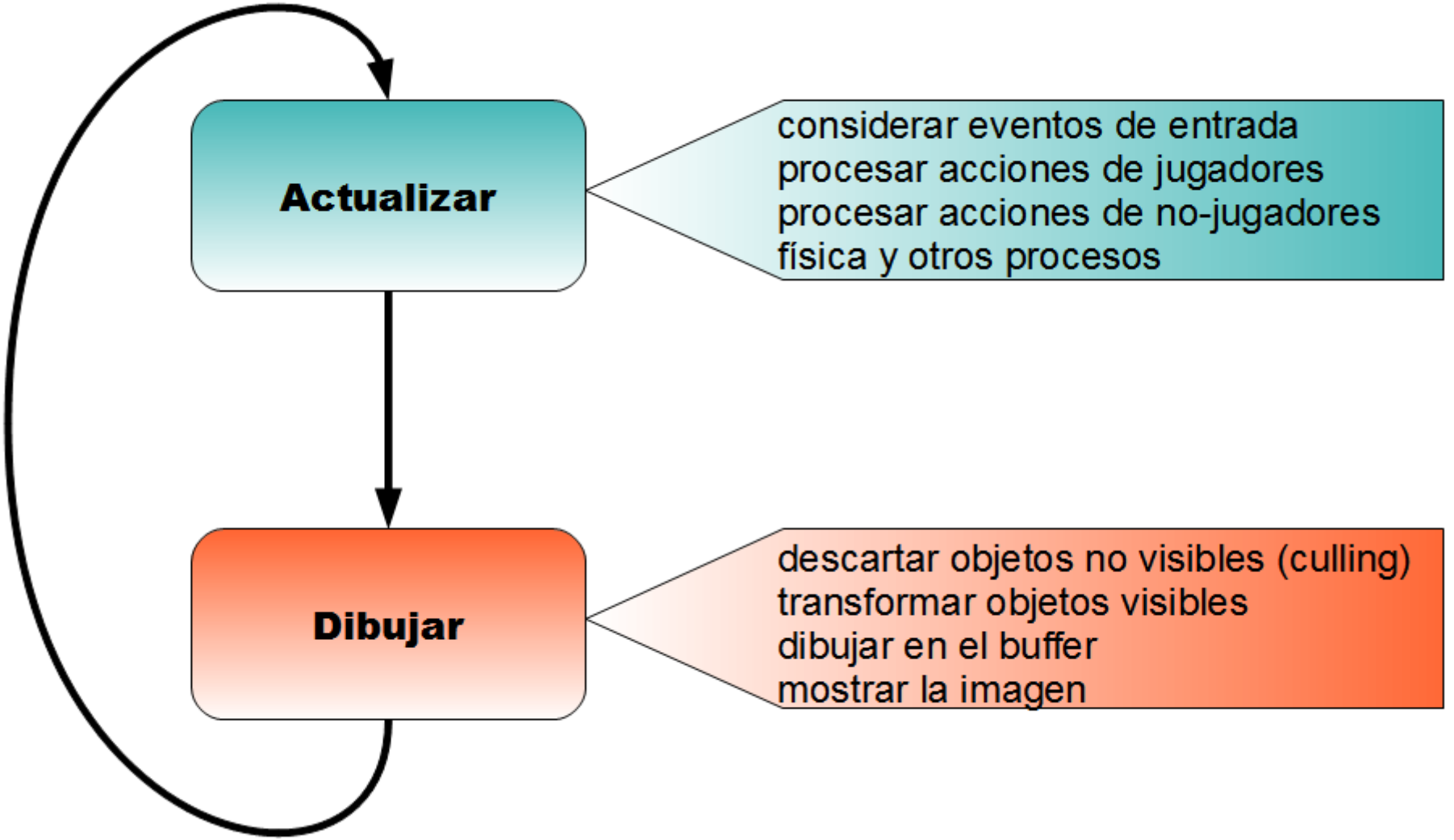
2.2 Componentes



2.2 Componentes

- **Manejador de eventos de entrada:** maneja los eventos de entrada y los pone a disposición del motor del juego, capas y objetos.
- **Motor del juego:** se encarga de manejar, actualizar y dibujar capas.
- **Capas:** se encargan de construir, actualizar y dibujar sus objetos.
- **Objetos:** elementos del juego como pueden ser, personajes, elementos del universo, fondo del juego, etc.
- **Manejadores de gráficos/sonidos/etc**: encargados de cargar los gráficos/sonidos y demás datos necesarios en el juego.

2.3 Main Loop


Actualizar
considerar eventos de entrada
procesar acciones de jugadores
procesar acciones de no-jugadores
física y otros procesos
Dibujar
descartar objetos no visibles (culling)
transformar objetos visibles
dibujar en el buffer
mostrar la imagen

2.3 Main Loop

Actualizar

- Es la etapa donde se altera el **estado del juego**.
- El **estado del juego** es un conjunto de variables que mantienen la información de cada elemento del juego.
- Incluyen la información de cada objeto del juego, posiciones, valores de recursos, puntaje, variables globales, etc.

2.3 Main Loop

Eventos de entrada

- Los métodos de entrada tradicionales son **orientados a eventos**.
  El evento captura el estado del dispositivo.

- En el main loop se usa **polling** para la entrada.
  Se consulta el estado al comienzo del loop.
  Solo un evento en cada actualización del loop, genera pérdida de eventos.
  Los métodos orientados a eventos no tienen esos problemas.

2.3 Main Loop

Eventos de entrada

- Se pueden combinar métodos orientados a eventos con el polling para evitar las pérdidas, por ejemplo usando una cola de eventos.
- En cada etapa de actualización del loop se consumen los eventos ocurridos.
- Es de utilidad al menos tener un hilo para escuchar los eventos.
  - Esto trae aparejados problemas de mutuaexclusión. Puede ser costoso.

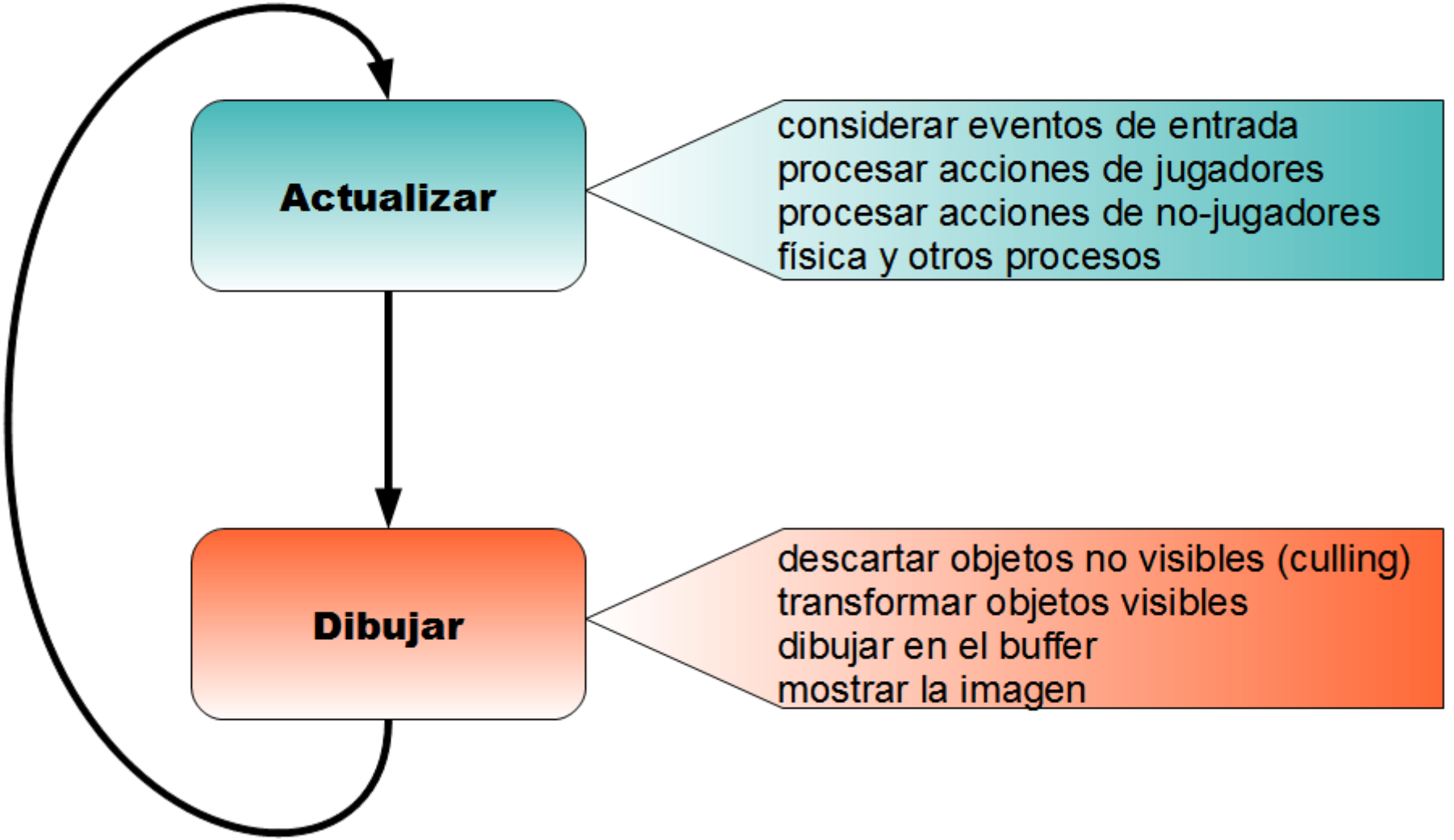
2.3 Main Loop

Eventos de entrada

¿Es necesario recordar todos los eventos?

- Los frames son cortos (60Hz~16,7ms).
- Generalmente la frecuencia de eventos es mayor al tiempo del frame.
- Generalmente las perdidas son tolerables.
- Procesar los eventos del buffer puede ser costoso en tiempo.

2.3 Main Loop

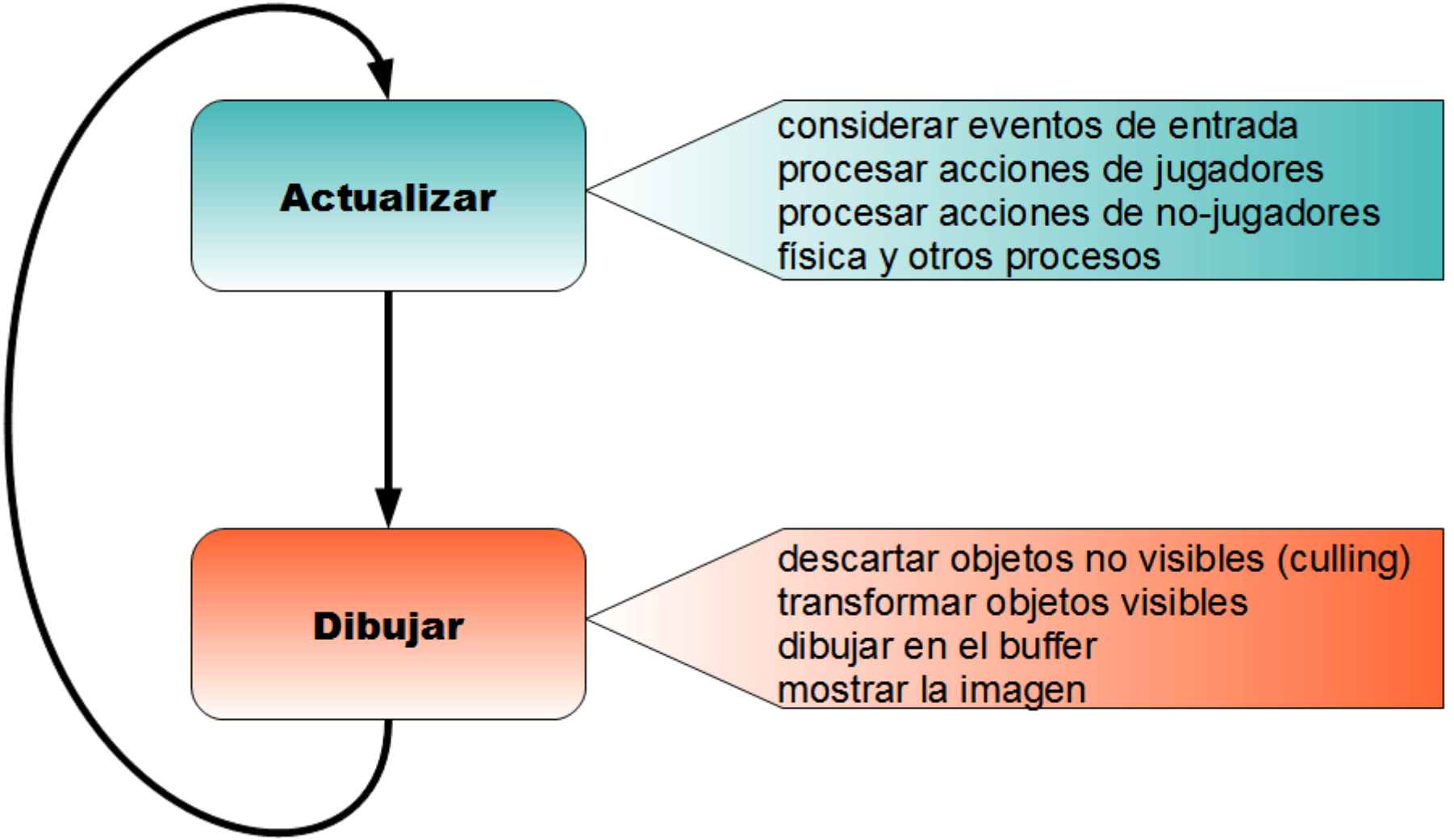

Actualizar
considerar eventos de entrada
procesar acciones de jugadores
procesar acciones de no-jugadores
física y otros procesos
Dibujar
descartar objetos no visibles (culling)
transformar objetos visibles
dibujar en el buffer
mostrar la imagen

2.3 Main Loop

Procesar las acciones de los jugadores

- Las acciones alteran el estado del juego.
- Dichas acciones se corresponden con los eventos de entrada.
- Se interpretan los valores de las variables de los dispositivos y se traducen en acciones.
  Mover, atacar, etc.
  Es útil tener funciones para cada acción.

2.3 Main Loop


Actualizar
considerar eventos de entrada
procesar acciones de jugadores
procesar acciones de no-jugadores
física y otros procesos
Dibujar
descartar objetos no visibles (culling)
transformar objetos visibles
dibujar en el buffer
mostrar la imagen

2.3 Main Loop

Procesar las acciones de los NO jugadores (NPC)

- Llamamos no jugadores a las entidades manejadas el computador.
- Tienen sus propias acciones pero no son controladas por entradas. Se utiliza **Inteligencia Artificial**.
- Cumplen con el siguiente ciclo.
  (1)Percibir: obtienen información útil de lo que los rodea.
  (2)Pensar: eligen una acción.
  (3)Actuar: actualizan el estado del juego.

2.3 Main Loop

Procesar las acciones de los NO jugadores

- El paso de Actuar por lo general es rápido.
  Si el juego se enlentence → solo actuar.
- Percibir y Pensar son pasos más complejos.

2.3 Main Loop

Procesar las acciones de los NO jugadores

Un problema común: Los “no jugadores” reaccionan muy rápido.

- Sucede como consecuencia de procesar sus acciones en forma secuencial.
- Una posible solución es que solo reaccionen a lo que ven, es decir, elegir una acción pero no realizarla.
- Otra es retrasar las acciones cierta cantidad de loops.

2.3 Main Loop

Procesar las acciones de los NO jugadores

Actuar sin pensar.

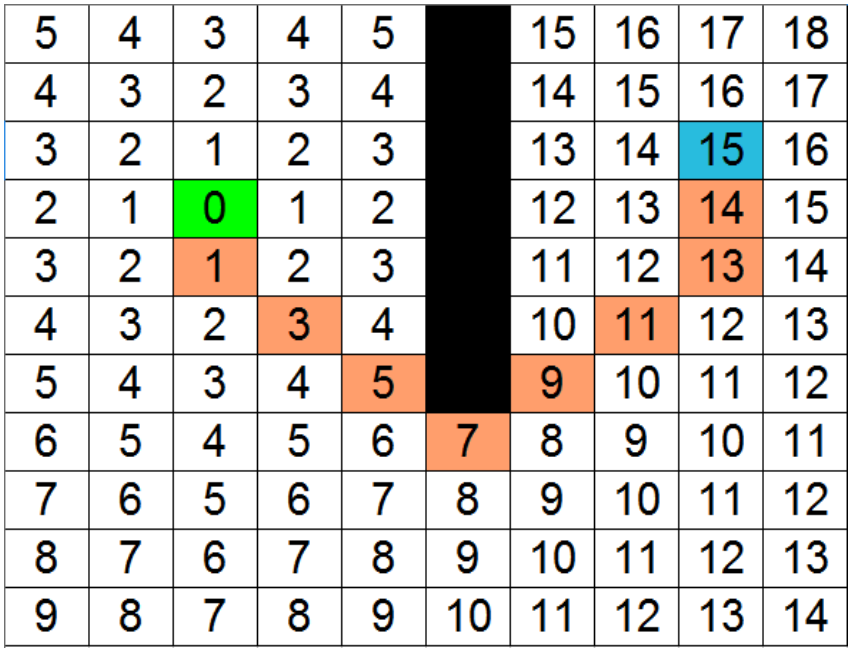
- Para esto se requiere recordar la última acción para poder seguir realizándola
- Se necesita guardar la acción y parámetros.
- Si se decide finalizar la acción se puede pensar para elegir la nueva acción. (elegir al azar podría ser una opción para no jugadores “tontos”)

2.3 Main Loop

Procesar las acciones de los NO jugadores

Encontrar el camino óptimo.

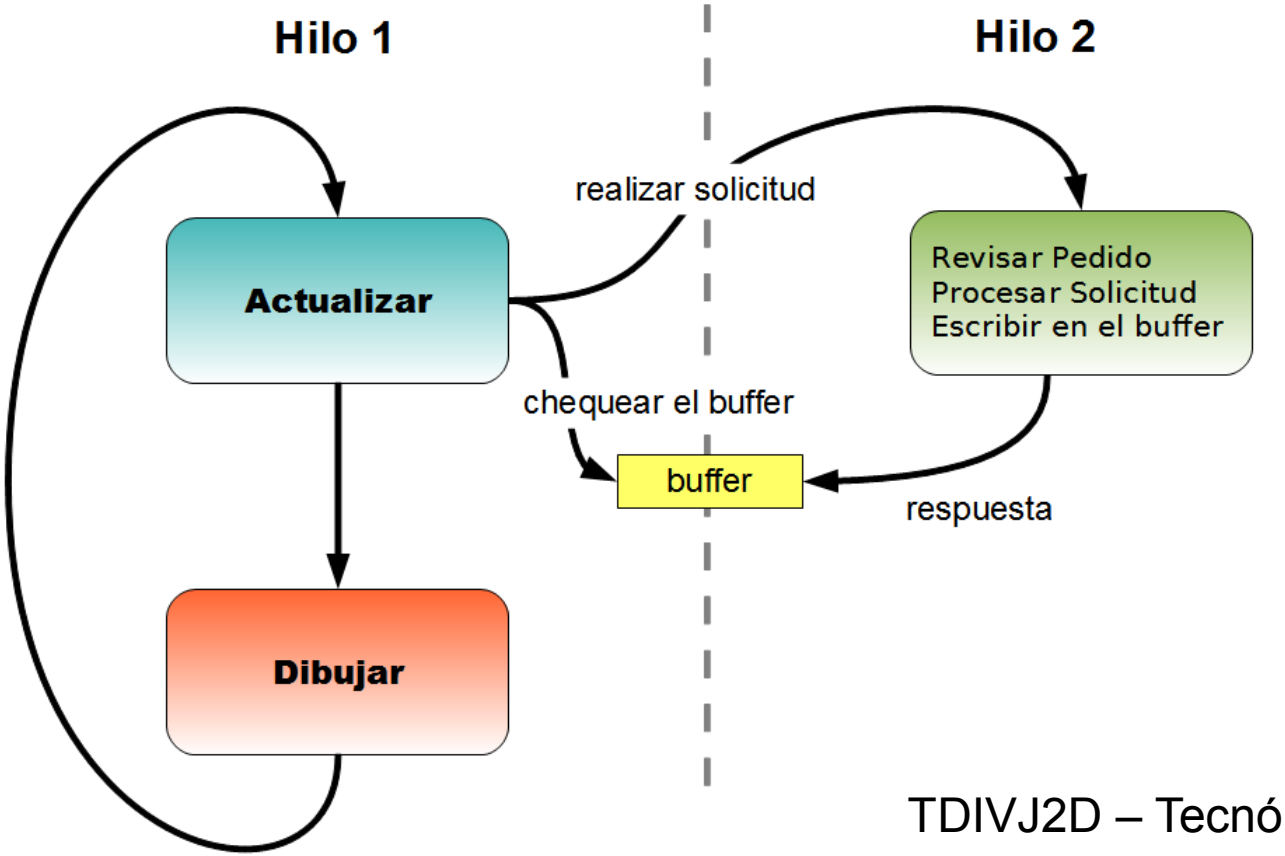
- Problema común, no siempre resuelto de la mejor forma.
  Es importante si el juego se visualiza desde arriba.
- Muy costoso en términos computacionales. (Dijkstra)



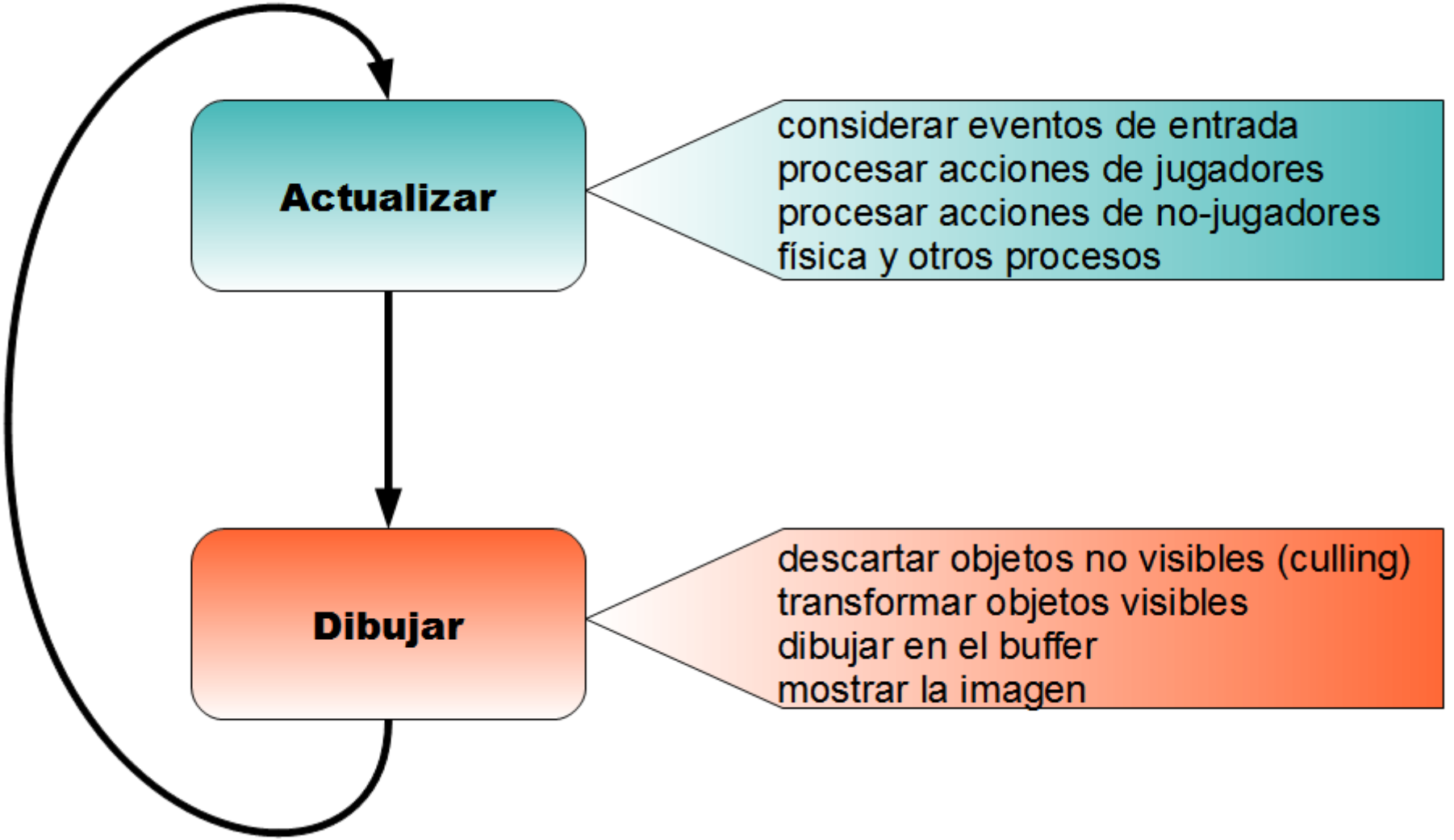
2.3 Main Loop

Procesar las acciones de los NO jugadores

Para los cálculos que llevan mucho tiempo se puede utilizar hilos.



2.3 Main Loop


Actualizar
considerar eventos de entrada
procesar acciones de jugadores
procesar acciones de no-jugadores
física y otros procesos
Dibujar
descartar objetos no visibles (culling)
transformar objetos visibles
dibujar en el buffer
mostrar la imagen

2.3 Main Loop

Física

¿Para que? → Mover los objetos sobre la pantalla.

- Cinemática: Sin tener en cuenta a las fuerzas externas.
- Dinámica: Teniendo en cuenta el efecto de las fuerzas.
- Colisiones entre objetos.
  Detección de colisiones: ¿Ocurrió una colisión?
  Resolución de colisiones: ¿Qué hacer?

2.4 Manejo del Tiempo

- Al principio se usaba el reloj del sistema. La velocidad del juego dependía directamente de la velocidad del procesador.
- La siguiente aproximación fue:

```
setTickerEvent ( 30/s )
when( tickEvent )
{
    update()
    render()
}
```

Las actualizaciones por segundo disminuían si los tiempos de update y render demoraba mucho. Por lo que el juego se enlentecía.

2.4 Manejo del Tiempo

Manejo del tiempo hoy

- Suponiendo que tenemos los siguientes métodos:
  Update (): actualizar todos los objetos del juego.
  Render (): dibuja todos los objetos del juego.
  Time (): obtiene la hora actual en ms.
  Sleep (ms): suspende por ms cantidad de ms.
- Y la siguiente restricción y objetivo:
  Ups - meta de actualizaciones por segundo (restricción).
  Fps = Ups (objetivo)

2.4 Manejo del Tiempo

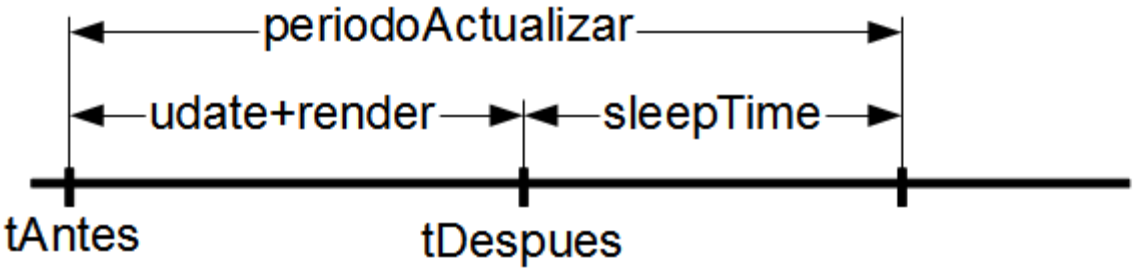
Manejo del tiempo hoy

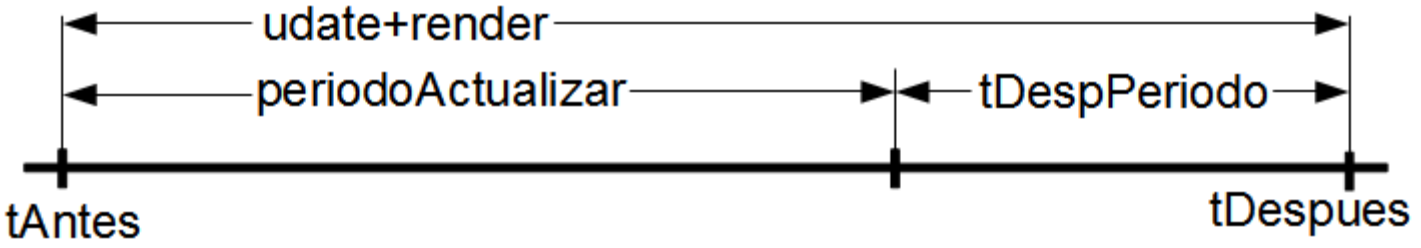
periodoActualizar = 1000 / UPS
faltante = 0 /*tiempo excedido del período en el loop anterior*/
tDespPeriodo = 0 /*tiempo excedido del período cuando es mayor que periodoActualizar*/

```
while( running ) {
    tAntes = time()
    update()
    render()
    tDespues = time()
    sleepTime = periodoActualizar - (tDespues - tAntes) - faltante
    if( sleepTime > 0 ) {
        sleep(sleepTime)
        faltante = time() - tDespues - sleepTime
    }
    else {
        tDespPeriodo = - sleepTime
        faltante = 0
        while( tDespPeriodo > periodoActualizar ) {
            update()
            tDespPeriodo = tDespPeriodo - periodoActualizar
        }
    }
}
```

2.4 Manejo del Tiempo

Manejo del tiempo hoy





2.5 Elementos de Diseño

Temas a definir:

- Decisiones de alto nivel la arquitectura
  ¿Cual es la arquitectura a seguir?
  ¿Qué bibliotecas se utilizarán?

- División del trabajo entre el equipo
  ¿Quién trabaja en qué parte del juego?
  ¿Qué es lo que necesitan para coordinar?

2.5 Elementos de Diseño

Definir **Subsistemas**

- Son una unidad lógica de funcionalidad.
- A menudo reutilizables en múltiples juegos.
- Los detalles de implementación se ocultan.
- API describe la interacción con el resto del sistema.
- Forma natural para descomponer el trabajo.
- Cada programador podría decidir como implementar, pero todo el equipo debe ponerse de acuerdo sobre la API.

2.5 Elementos de Diseño

Especificar **Clases**, **Responsabilidades**, **Contratos**.

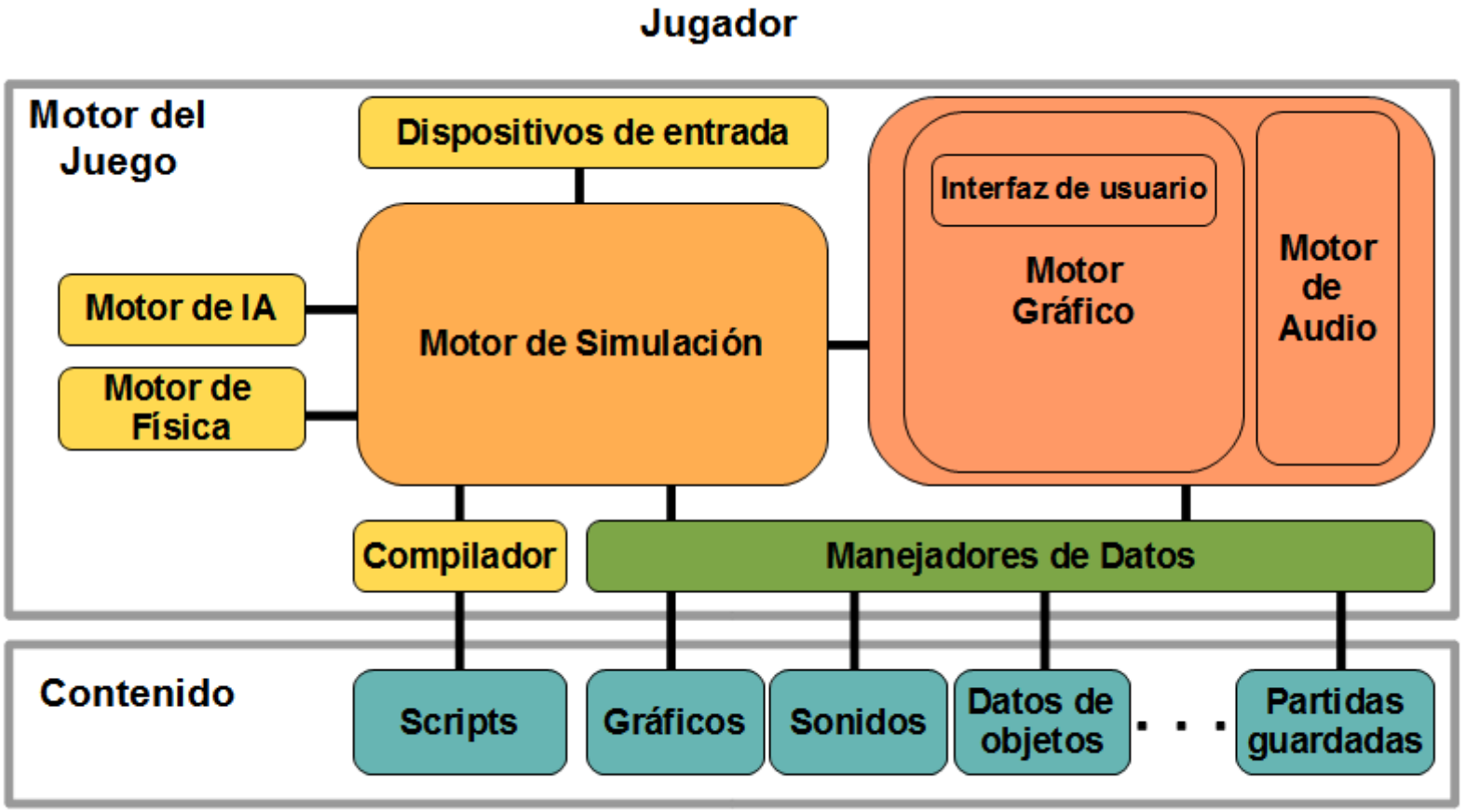
- Clases: elementos importantes en el subsistema.
- Responsabilidades: lo que hace la clase.
- Contratos: especificación de los métodos.
  ¿Qué parámetros?
  ¿Qué valores se devuelve el método?
  ¿Qué hace el método?
- Colaboraciones: interacción entre clases para lograr un resultado.
  Las colaboraciones que se dan dentro de un subsistema pueden obviarse en procesos de desarrollo ágiles.
  Son útiles los diagramas de flujo y de actividad para los procesos de actualización del loop.

2.5 Elementos de Diseño

Contrato de programación:

- Una vez que se crea el API, es un contrato.
- Promesa del equipo de que "trabaja de esta manera".
- Evitar cambios de interfaz.

2.6 Ejemplo de Arquitectura



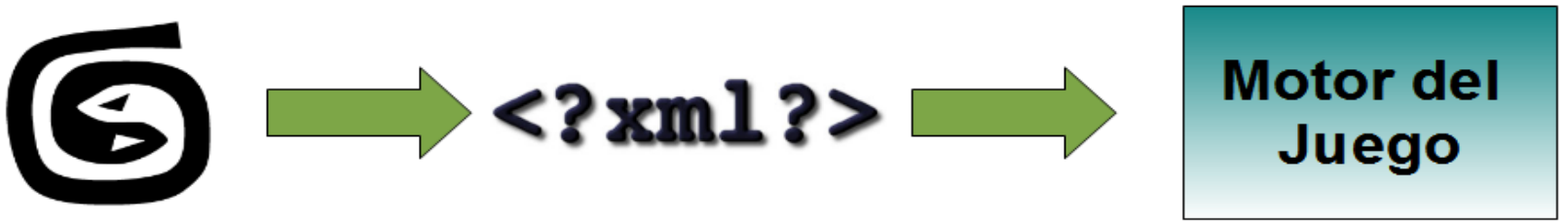
2.7 Data Driving

- No hay código fuera del motor.
- El motor determinará hasta donde es posible.
- Por ejemplo:
  Arte y música en formatos estandar.
  Información de objetos en formato XML u otros formatos.
  Interfaz de usuario especificada en XML u otros formatos.
  Comportamiento de personajes especificado en scripts.

2.7 Data Driving

¿Porqué usar Data Driving?

- Programadores crean el motor del juego, enfocándose en el desarrollo tecnológico.
- Los diseñadores crear el contenido del juego. Por lo general el contenido artístico / comportamiento.
- Data Driving optimiza el proceso de producción.
- Se puede extender la vida útil del juego manteniendo los contenidos frescos.

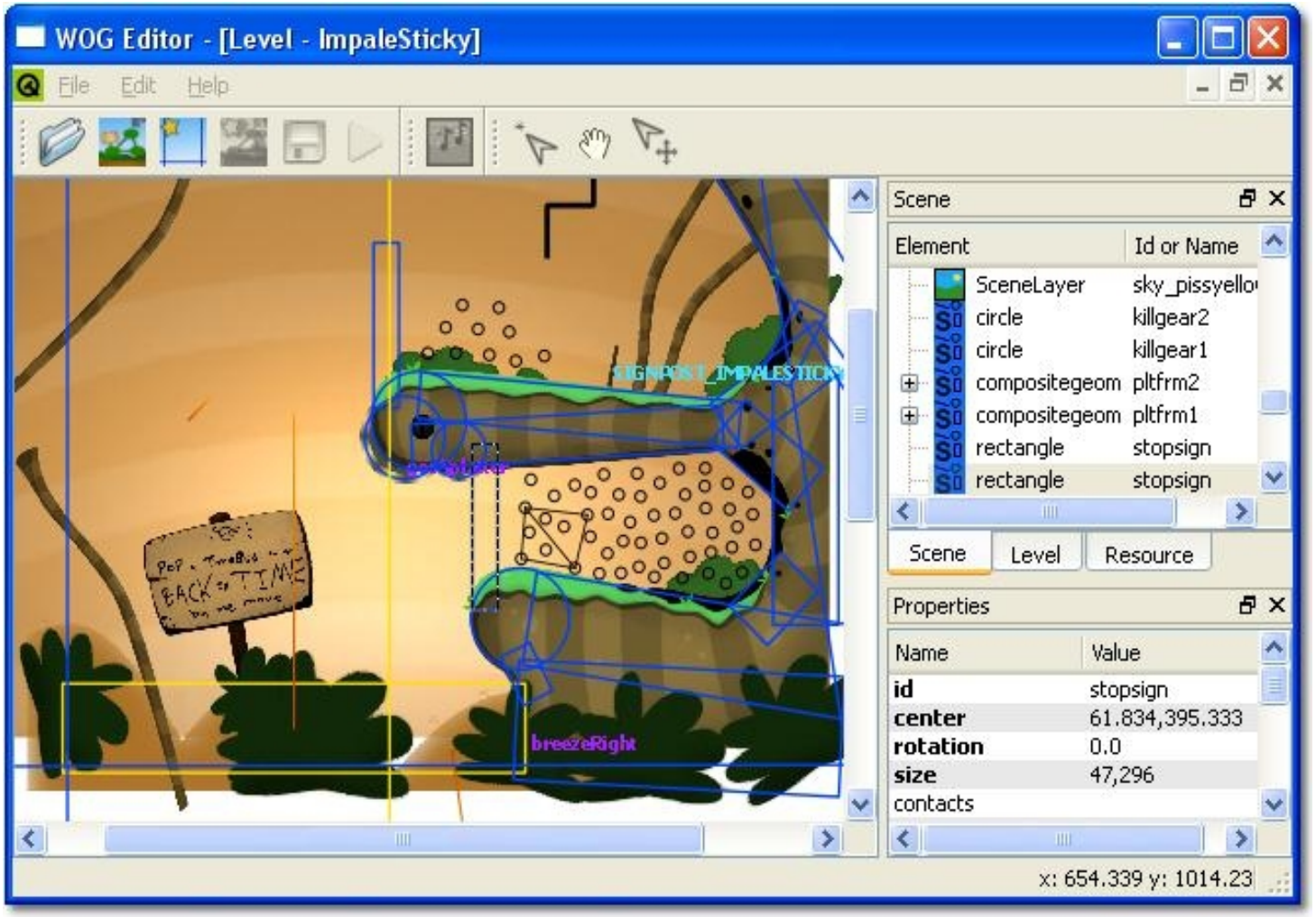


2.7 Data Driving

Ciclo de vida de Data Driving

- Comienza con un pequeño número de programadores.
- Los programadores crear un “content pipeline”: Herramientas de producción para los artistas y diseñadores. Los datos se pueden importar, visualizar y probar.
- Se aumenta la cantidad de artistas y diseñadores para creación de contenido.
- Se busca un nuevo título y se comienza de nuevo.

2.7 Data Driving



2.7 Data Driving

Herramientas de creación de contenido.

Editor de niveles:

- Crear desafíos y obstáculos.
- Colocar los objetos en el mundo.
- Ajustar parámetros (física, dificultad, etc)

Lenguajes de Scripting

- Definir el comportamiento de carácter.
- Activadores de guión y eventos.
- Diseño de interfaz de usuario.

Referencias

- Los cuatro tipos de Bartle (http://www.mud.co.uk/richard/hcds.htm)
- Curso Java Games Programming by Philip Hanna - Queen's University, Belfast, Ireland
- The Game Design Initiative at cornell university - http://gdiac.cis.cornell.edu/
- Andrew Rolling, Dave Morris - Game Architecture and Design. 2003
- Richard Rouse - Game Design Theory and Practice. 2005


El traitro
100
Usul
53
El infamo
63
El grotesko
16