***1.1. Arquitectura de juego***

Un juego es una aplicación que recibe eventos de entrada por parte del jugador, realiza un proceso lógico en función del argumento del juego y genera gráficos junto con sonidos que representan su estado actual.

***1.1.1. Entrada del usuario***

Una forma de gestionar los eventos de toques de pantalla es definiendo áreas de control. En función de la posición donde se toque dentro de cada área de control el motor del juego realizará una determinada acción.

***1.1.2. Lógica del juego***

El módulo de la lógica del juego es el responsable de cambiar el estado de los actores en el juego. Un actor es un objeto que tiene un estado en el juego. Dentro de la lógica del juego también se encuentra la detección de colisiones, los movimientos con físicas, motores de mapas de pantalla y las acciones inteligentes y coherentes de los actores.

***1.1.3. Sonido***

El módulo de sonido produce los efectos sonoros en función del estado actual en que se encuentre el juego. Casi todos los actores del juego producen algún tipo de sonido en sus distintos estados. Dependiendo del tipo de dispositivo los canales de sonido pueden ser muy limitados en cuanto a la cantidad de sonidos que pueden emitirse a la vez. El motor de juego debe administrar los canales y decidir qué sonido producir en cada momento. Todos los sonidos tienen que estar sincronizados con los movimientos y acciones de los actores.

***1.1.4. Gráficos***

Este módulo se encarga de representar el estado del juego en la pantalla. Se dibuja en el lienzo que representa la vista del juego o se tiene una memoria gráfica separada que se le pasa a la vista. La vista puede ser una concreta de OpenGL o una personalizada. La representación de los gráficos se mide en cuadros por segundos / FPS; este indica las veces por segundo que se representa el estado del juego. Cuantos más FPS se tenga, se obtendrá un movimiento más suave y fluido de los actores en pantalla; esto requiere de mayor potencia de cálculo.

***1.1.5. Salida***

La salida es el resultado de la combinación de sonidos, imágenes y algún movimiento vibtatorio, consiguiendo que el jugador obtenga una combinación de sensaciones coherentes con los eventos de entrada que genera, con una lógica esperada.

***1.2. El bucle principal del juego***

Para que la arquitectura del juego funcione correctamente es necesario tomar los valores de la entrada, actualizar el estado interno según su lógica y representar en pantalla y mediante sonidos el nuevo estado del juego; todo ello se realiza de manera cíclica y rápida. Estas tres tareas deben llevarse a cabo de forma explícita/manual o implícita/motor de juego.

***1.3. Motores de juegos***

Los sistemas interactivos usan los sistemas de entrada y salida a gran velocidad, esto permite construir APIs de alto nivel que facilita el desarrollo como un framework gráfico orientado a juegos hasta motores completos de juegos con gestión gráfica.

Las partes de un motor de juego son:

1. Percepción: El actor simula sensores que le permiten detectar objetos o propiedades que le interesen.
2. Toma de decisión: En función de los valores percibidos deberá decidir qué acción realizar.
3. Movimiento: Determina la dirección y camino a tomar para acercarse al objetivo, evitando los obstáculos que se encuentren.

***2. Sprites***

Los sprites son componentes básicos en los juegos 2D. Estos son combinaciones de una imágen gráfica y su información geométrica de posición y rotación. Su parte gráfica viene dada por una textura (imagen descomprimida y guardada en la memoria del procesador gráfico) o región de textura. Para visualizar una textura se usa una figura, normalmente un rectángulo para describir la geometría donde se va a realizar la proyección / mapping.

La proyección de textura o mapeo de texturas es un proceso que calcula en qué parte del espacio y con qué forma dibujar la textura.

***3. Motor de físicas***

Todos los efectos como explosiones, sangrados, desplomes, etc están relacionados con las propiedades físicas que representan los actores del juego, sus herramientas y sus accesorios. Todos los juegos tienen un mínimo de simulación física, aunque hoy en día existen una gran variedad de efectos físicos que han sido generados por motores físicos.

Un motor físico es un sistema software que simula propiedades físicas en cuerpos sólidos, líquidos o dinámica de fluidos. Estos motores están formados por tres módulos:

1. Módulo de simulación dinámica: Aplica las leyes físicas a los elementos del sistema en tiempo real y con una buena percepción del resultado.

1. Módulo de colisiones: Detecta y resuelve las colisiones entre los elementos / actores del juego.
2. Módulo de representación de sistemas: Donde se guardan los elementos que serán modelados para realizar cálculos. Este depende del tipo de simulación realizada.

Algunas características de los motores de físicas son:

1. Colisiones: Métodos de rellamada para contactos, trabajar con polígonos y círculos, figuras compuestas por figuras más simples y grupo de colisiones y categorías.
2. Simulación física: Física continua con resolución de tiempo de impacto, grafos de cuerpos conectados, contacto, fricción y restitución, apilado estable con resolución de tiempo lineal y rotura de objetos conectados según sus propiedades físicas.
3. Sistema: Pequeño tamaño de los bloques y de la pila, parámetros centralizados para afinar y alta portabilidad.

Los motores de juegos como Unity incorporan de forma automática muchos elementos propios de un motor de físicas que ahorran trabajo, aunque es necesario ajustar sus parámetros al proyecto.

***4. Detectar colisiones***

La detección de colisiones es una funcionalidad muy importante dentro de los juegos, esto nos permite: Que los sprites no salgan de los límites, que no se atraviesen obstáculos sólidos, que se detecte cuando se toca otro sprite del juego o cuando un proyectil colisiona con otro sprite. Esta se lleva a cabo en dos pasos:

1. Colisión gruesa: Determina que dos figuras probablemente van a colisionar. Se implementa mediante una malla de celdas cuadradas donde cada figura pertenece a la celda de la malla que contiene su centro.
2. Colisión fina: Determina el resultado de la colisión con las dos figuras anteriores. Para ello se usa como base las matemáticas geométricas, trigonométricas y algebraicas. Su cálculo se realiza mediante pertenencias de estas ramas de matemáticas como: Intersecciones en rectas y planos, entre curvas y rectas o pertenencias.

La detección de colisión mediante proyecciones se realiza mediante la técnica de proyección. Este método detecta el solapamiento entre objetos, la idea básica es mover los objetos al punto en el que hay un mínimo de solapamiento posible, de tal forma que se resuelve la colisión encontrando al vector que mueve a los dos objetos a un mismo punto de solapamiento mínimo. Para que no se solapen los objetos también se usa la técnica basada en penalización por fuerza; este usa muelles para empujar un objeto a que se separe del otro.

El teorema de separación de ejes dice que si tenemos dos figuras convexas y encontramos un eje en el que la proyección de las figuras no se solape, las figuras no chocan entre sí. Esta técnica es muy usada en el 2D ya que reduce el cálculo en 1D.

El teorema de separación de ejes para AABB es una figura muy usada en juegos 2D. El uso de este tipo de figuras simplifica el cálculo de colisiones con los ejes X e Y sobre otras figuras.

El problema de los movimientos rápidos es que cuando se realiza el cálculo de colisiones y se acercan los objetos, ambos van a gran velocidad en comparación con su tamaño. Esto puede ocasionar que en el proceso de actualización el incremento de coordenadas sea tan grande que ambos objetos se salten sin colisionar. Para ello hay dos soluciones:

1. Colisión con test de barrido: Realiza un barrido de la trayectoria formada por la recta que une los dos puntos a actualizar en el movimiento (punto origen y punto final del sprite). Es una técnica efectiva pero aumenta mucho el número de cálculos.
2. Sobre muestreo: En cada actualización incrementamos el tiempo dando pequeños pre-incrementos. Si normalmente se incrementa un segundo en cada actualización, ahora se actualizará cada medio segundo antes de pintar, esto ayudará a detectar colisiones. Esta técnica es menos efectiva pero no requiere muchos cálculos y el sistema no cambia.

**5. Motor de escenas**

Los juegos se desarrollan dentro de un escenario o imagen de fondo que no actúa en el juego, simplemente se usa para adornar. La tarea de crear estos escenarios es sencilla y automatizable, para ello se han definido varios estándares de definición de escenas: El formato TMX basado en XML para juegos 2D permite crear mapas de baldosas con proyección isométrica. El programa más usado es Tiled Map Editor.

El funcionamiento de estas aplicaciones es sencillo: Se crea uno o varios conjuntos de texturas con sus regiones, se cargan las texturas; se definen las capas dentro del mapa, una capa de obstáculos y otra de fondo; se distribuyen las regiones de texturas en las capas que corresponda y por último se guarda el mapa.tmx para cargarlo en el juego.

***6. Motor de inteligencia artificial***

La inteligencia artificial en los juegos hace referencia a las técnicas para producir la ilusión de inteligencia en el comportamiento de los actores del juego, como los enemigos. Estas técnicas se basan en métodos de inteligencia artificial. Su objetivo es proporcionar la funcionalidad de esta inteligencia a los juegos independientemente del resto de elementos del juego como la lógica, el motor gráfico o la renderización.

***7. Motor de sonidos***

Los motores de sonido simplifican la tarea de realizar sonidos en el juego. Hay dos tipos de sonidos:

1. Sonidos melódicos: Se suelen escuchar de fondo, como una canción, son opcionales y pueden desactivarse.
2. Efectos sonoros: Cada elemento/actor tendrá uno o varios sonidos asociados.

***8. API de gráficos en 3D***

La librería, API o motor gráfico por excelencia en Android y otras plataformas es OpenGL 1.0 y 2.0. Este usa un sistema de coordenadas tridimensional donde el eje z está frente a la pantalla con valores positivos alejándose de esta. Las figuras en OpenGL se denominan primitivas y están formadas por puntos que definen los vértices / vertex. OpenGL no usa píxeles para las coordenadas, sino que hace uso de valores en números reales codificados con números flotantes de 32 bits. El tamaño de la imagen dependerá de las propiedades de proyección, del puerto de visión y de las propiedades de la cámara. OpenGL añade primitivas para la gestión de memoria, estas trabajan con la memoria gráfica que accede al procesador gráfico que es más rápido. Si se usa OpenGL para programar se debe dedicar muchas instrucciones a reservar y liberar este tipo de memoria, por lo que es más enrevesado de programar aunque da mucha potencia y velocidad a los gráficos.

Los vértices están unidos de dos a dos, esta unión se da por el orden en que se definen. Estos tienen dos caras, de ellas solo se visualiza la delantera.