Tarea 4: Resumen de Vídeos, Desarrollo de Videojuegos en 8 bits

Nombre: Vivian Marino Sánchez

Código: 62124

Docente: José Laruta

1. Resumen

La programación de gráficos para NES (Nintendo Entertainment System) puede ser compleja. Esta programación es en 8 bits, esto quiere decir que todo lo programado en NES tiene un valor entre 0 y 255, únicamente. Para entender cómo se obtiene la parte gráfica, es necesario primero entender el hardware de NES: la PPU (Picture Processing Unit). Es un chip especializado diseñado por Nintendo que funciona como hoy en día lo hacen las tarjetas gráficas. Tiene 2KB de sistema RAM y 2KB para video RAM, no tiene guardado permanente y tiene secciones de solo lectura: 32KB de código y 8KB de datos de gráficos. No se puede programar directamente en ella. Tiene una memoria RAM que puede ser modificada para cambiar cómo se generan los gráficos. Esta memoria está dividida en 4: la primera contiene las tablas de patrones, donde se guardan los sprites, luego vienen las tablas de nombres donde se encuentran múltiples composiciones basadas en un mapa de todos los bytes de la pantalla, a continuación tenemos la paleta de colores donde se guardan hasta 8 paletas (4 para fondos y 4 para foreground) que contienen 4 colores cada una, a pesar de que la PPU puede producir hasta 50 colores, por último se tiene a la OAM (Object Attribute Memory) que es donde se controlan los elementos del foreground de la pantalla.

La PPU mandaba sus gráficos a televisores por rayos catódicos. Estos televisores contenían un tubo de rayos catódicos. Este tubo dispara electrones por medio de rayos (si era en blanco y negro por medio de uno solo, si era a color por medio de 3: rojo, azul y verde). En cuanto estos electrones llegan a una parte de la pantalla, esta se enciende con el color o colores que formen la cantidad de electrones que le lleguen de cada rayo. Para generar una imagen completa, los tubos de rayos catódicos recorren toda la pantalla y repiten el proceso mientras el televisor esté encendido. La PPU enviaba señal a los televisores de rayos catódicos para indicar la intensidad de cada rayo. Esta señal podía ser de dos tipos: NTSC (EEUU y Japón) o PAL (Europa y Sudamérica), donde el segundo tipo es más lento.

Para generar los gráficos que serían envíados al televisor, cada casilla de las tablas de patrones contenían como máximo la clave de 4 colores de alguna de las paletas activas. La manera en que estas paletas funcionan es la siguiente: el primer color definirá la transparencia y los 4 colores están numerados de 0 a 3, de tal manera que cada color está compuesto por 2 bits, así cada casilla de una tabla de patrones tiene un peso de 16 bytes. Las tablas de patrones juntas pesan 8KB. Aunque decimos que los colores se guardan en 2 bits, en realidad se guardan en dos tablas separadas: la tabla low bit y la tabla high bit. Estas tablas serán unidas para obtener el color real de la casilla. A continuación en las tablas de nombres cada casilla hace referencia a una casilla en alguna tabla de patrones. Cada pantalla generada por una tabla de nombres

ocupa 1 byte por casilla. Toda pantalla generada reutiliza recursos de la tabla de patrones para ahorrar espacio. La PPU carga a la pantalla lo que hay en las tablas de nombres gracias a las tablas de atributos asociadas a ellas. Estas tablas de atributos contienen en cada casilla las paletas de colores que corresponden a cada cuarta parte de una casilla en una tabla de nombres para cargar de manera correcta los colores. Por último, la OAM almacena hasta 64 sprites al mismo tiempo y cada uno tiene 4 bytes: la coordenada vertical, la tabla de patrones que le corresponde, los atributos que corresponden al sprite y la coordenada horizontal. En los atributos los primeros dos bits definen la paleta de colores, no se usan los bits 2,3 y 4, el 5to bit define si el sprite irá delante o detrás del fondo y los bits 6 y 7 definen si el sprite se invertirá horizontal o verticalmente. Entonces, podemos construir la parte gráfica de nuestro juego en capas.

En cuanto a la parte de mecánica del juego, la PPU solamente puede sumar, dividir, multiplicar por dos y dividir por dos. Entonces, para hacer cualquier otra operación, se deben programar funciones. Para simular la física del mundo real, es necesario abstraer y simplificar la operación que tiene que hacerse para no ocupar espacio innecesario y realizar la tarea de la mejor manera posible. Tampoco se tiene un generador de números aleatorios. Para esto último se puede usar un LFSR (Fibonacci linear feedback shift register): hacer un XOR con los bits 1 y 9, almacenarlos en el bit 16 y hacer un shift a la derecha., o se puede hacer una tabla con números predeterminados.

Para guardar un juego al principio se tenía un sistema de contraseñas. Estas contraseñas guardaban las partes que eran necesarias para volver a cargar una partida. Después, Nintendo creó disquetes para guardar los datos sin tener que usar un sistema demasiado complejo para el usuario. Estos disquetes todavía no controlaban que la información no se corrompa por errores del usuario o de la consola. Entonces se empezó a guardar datos en múltiples copias para compararlas y evitar la pérdida de datos.

2. Puntos clave

La PPU está dividida en 4 partes: la primera contiene las tablas de patrones, donde se guardan los sprites, luego vienen las tablas de nombres donde se encuentran múltiples composiciones basadas en un mapa de todos los bytes de la pantalla, a continuación tenemos la paleta de colores donde se guardan hasta 8 paletas (4 para fondos y 4 para foreground) que contienen 4 colores cada una donde el primer color define la transparencia, por último se tiene a la OAM (Object Attribute Memory) que es donde se controlan los elementos del foreground de la pantalla.

Para generar los gráficos que serán envíados a la pantalla, cada casilla de las tablas de patrones contienen como máximo la clave de una paleta y referencia los 4 colores que le corresponden. La manera en que estas paletas funcionan es la siguiente: el primer color definirá la transparencia y los 4 colores están numerados de 0 a 3, de tal manera que cada color está compuesto por 2 bits. Los colores en realidad se guardan

en dos tablas separadas: la tabla low bit y la tabla high bit. Estas tablas serán unidas para obtener el color real de la casilla. A continuación en las tablas de nombres cada casilla hace referencia a una casilla en alguna tabla de patrones. Toda pantalla generada reutiliza recursos de la tabla de patrones para ahorrar espacio. La PPU carga a la pantalla lo que hay en las tablas de nombres gracias a las tablas de atributos asociadas a ellas. Estas tablas de atributos contienen en cada casilla las paletas de colores que corresponden a cada cuarta parte de una casilla en una tabla de nombres para cargar de manera correcta los colores de cada cuadrante. Por último, la OAM almacena hasta 64 sprites al mismo tiempo, donde cada sprite es en realidad tan solo una casilla y cada uno tiene 4 bytes: la coordenada vertical, la tabla de patrones que le corresponde, los atributos que corresponden al sprite y la coordenada horizontal. En los atributos los primeros dos bits definen la paleta de colores, no se usan los bits 2,3 y 4, el 5to bit define si el sprite irá delante o detrás del fondo y los bits 6 y 7 definen si el sprite se invertirá horizontal o verticalmente. Entonces, podemos construir la parte gráfica de nuestro juego en capas y reutilizando sprites si son simétricos.

3. **Opinión: ¿8 bits o modernos?**

Es innegable que hoy en día es un trabajo mucho menos tedioso y complejo el de realizar la parte gráfica de un videojuego. Si bien todavía la codificación de ciertas características que pueden ser únicas para un proyecto, existen librerías de física, sprites, funcionalidades, escenarios, etc. Todavía toma tiempo la codificación de un videojuego, pero ahora se tiene el espacio suficiente en servidores para renderizar los

gráficos con mayor rapidez y, por lo tanto, poder ponerles más detalles y testearlos. Es por esta evolución en el desarrollo de videojuegos que hoy en día tenemos títulos con gráficos sumamente realistas y detallados que permiten una mayor inmersión en la historia que están contando. Sin embargo, a mi parecer, es esta habilidad de tener gráficos impresionantes la que distrae muchas veces a los usuarios y a los desarrolladores. Aunque mecánicas buenas y física realista son aspectos loables, no se debe dejar de lado la experiencia del usuario y la calidad de la historia o de las mecánicas de juego. En mi opinión personal, existen juegos que se promocionan enteramente por la calidad de sus gráficos y la optimización de sus funcionalidades sin que ofrezcan algo nuevo a la franquicia a la que pertenecen o al género del que forman parte. No culpo a los fanáticos por comprar videojuegos por la calidad de los objetos y personajes, pero el mercado está saturado de títulos que son imitaciones de sus versiones anteriores y no innovan en conceptos o mecánicas. Los juegos de antes, por la complejidad de programación y falta de espacio no podían ofrecer funcionalidades que los de hoy en día sí pueden ofrecer, pero cada componente era innovador, único y hecho con mucho esfuerzo y pasión.