UNIVERSIDAD NACIONAL DE HURLINGHAM
Programación de Videojuegos 1
Profesor Facundo Saiegh

Objetivo del Cuatrimestre:

El propósito de este cuatrimestre es introducir a los estudiantes en los conceptos fundamentales de JavaScript y PixiJS, y familiarizarlos con el entorno de desarrollo de Google Chrome.
A lo largo del curso, los estudiantes adquirirán las habilidades necesarias para desarrollar un videojuego 2D utilizando la librería PixiJS.

Trabajo Práctico Final:

Organizados en grupos de dos personas, los estudiantes deberán desarrollar y presentar un videojuego 2D que cumpla con los siguientes requisitos:

1. **Cantidad de Personajes**: El juego debe incluir un número significativo de personajes (más de 50) que implementen el algoritmo Boids o una variación similar.
2. **Interactividad**: El usuario debe tener la capacidad de interactuar con el juego en tiempo real mediante el uso del mouse y/o teclado.
3. **Ejemplos de Juegos**: Algunas posibles ideas incluyen:
    - Juego de matar hordas de zombis.
    - Un perro que arrea ganado.
    - Simulador de pesca.
    - Pelea entre grandes grupos de personas.
    - Defender a una celebridad de sus fanáticos enloquecidos.

Requisitos Técnicos:

El juego desarrollado deberá incluir, como mínimo, los siguientes aspectos técnicos:

- **Input del Usuario**: Interacción a través de mouse y/o teclado.
- **Spatial Hashing**: Optimización espacial para manejar grandes cantidades de personajes.
- **Spritesheets**: Uso de múltiples spritesheets por personaje.
- **Algoritmo Boids**: Implementación del algoritmo Boids para la simulación de comportamiento grupal.
- **Movimiento de Cámara**: Movimiento suave de cámara utilizando interpolación lineal (Lerp).
- **Finite State Machine (FSM)**: Implementación de una máquina de estados finitos para gestionar las transiciones y comportamientos de los personajes.

Elementos Opcionales:

El juego puede también incorporar uno o más de los siguientes elementos opcionales, que serán abordados en clase, aunque con menor profundidad que los elementos obligatorios:

- **Perlin Noise**: Aplicación del ruido Perlin en algún aspecto del juego.
- **Técnicas de Optimización Algorítmica**: Aplicación de optimizaciones avanzadas en el algoritmo o simulación.
- **Motor de Física 2D**: Implementación de un motor de física como MatterJS, Box2D, PlanckJS, entre otros.
- **Serialización**: Capacidad de guardar y cargar partidas.

- **Flowfield**: Implementación de campos de flujo para guiar el movimiento de los personajes.
- **Path Finding**: Algoritmos de búsqueda de caminos, como A* o similares.

Estructura de las Clases:

Cada clase del curso estará dividida en dos partes:

1. **Primera Parte:** Durante la primera mitad de la clase, se dedicarán a resolver dudas y brindar apoyo a los estudiantes en la implementación de los contenidos vistos en la clase anterior dentro de sus proyectos
2. **Segunda Parte:** En la segunda mitad de la clase, se presentarán nuevos contenidos que los estudiantes deberán implementar en sus proyectos a lo largo de la semana. Estos nuevos conceptos serán fundamentales para la progresión del desarrollo del juego y su complejidad.

Plan de Clases:

El curso está estructurado en 12 clases, cada una con un enfoque específico. A continuación, se detalla el contenido de las mismas:

1. **Conceptos Básicos de JavaScript**
    a. **Uso de las Herramientas de Desarrollo de Google Chrome**:
        i. **Pestaña Network**: Monitoreo de solicitudes de red.
        ii. **Pestaña Sources**: Exploración del código fuente y manejo de archivos.
        iii. **Consola**: Ejecución de comandos y debugging en tiempo real.
        iv. **Debugger y Breakpoints**: Uso de breakpoints para depurar código.
        v. **LocalStorage**: Almacenamiento local en el navegador.
    b. **JavaScript Básico**:
        i. **Tipos de Datos y sintaxis**: Introducción a JavaScript.
        ii. **Gameloop**: Concepto de bucle de juego y su implementación básica.
        iii. **setTimeout, setInterval, RequestAnimationFrame**: Uso de `requestAnimationFrame` para sincronizar la animación con la tasa de refresco de la pantalla.
    c. **Uso de ChatGPT para Generación de Código**: Introducción al uso de herramientas de inteligencia artificial como ChatGPT para asistencia en la programación.

2. **PixiJS y Carga de Imágenes**
    a. **Introducción a PixiJS**: Presentación de la librería PixiJS y su importancia en el desarrollo de videojuegos 2D.
    b. **Panel de Desarrolladores de PixiJS**: Exploración del panel de herramientas de desarrollo específico de PixiJS.
    c. **Contenedores en PixiJS**: Creación y manejo de contenedores para organizar elementos gráficos.
    d. **Carga de Imágenes**: Métodos para cargar y mostrar imágenes en PixiJS.
    e. **CORS**: Explicación sobre las restricciones de CORS, introducción a servidores web y Node.js.
    f. **NPM y Servidores Locales**: Instalación y uso de algun paquete de Node para levantar un servidor local.

3. **Movimiento del Personaje con Teclado y/o Mouse**
    a. **Eventos en JavaScript**: Captura y manejo de eventos de teclado y mouse.
    b. **Gestión de Input**: Almacenamiento y procesamiento de la entrada del usuario dentro de la estructura del juego.
    c. **Asignación de Velocidad al Protagonista**: Implementación de movimiento basado en la entrada del usuario.
    d. **Uso de Spritesheets**: Introducción al uso de spritesheets para animar el personaje.
    e. **Herramientas Externas para el manejo de SpriteSheets**

4. **Entidades en Posiciones y Movimientos**
    a. **Creación de Clase Entidad**: Estructuración del código en una clase genérica de la cual heredan los demás tipos de objetos
    b. **Desarrollo de Clases para Personajes y Objetos**: Creación de clases específicas para el protagonista, NPCs, y objetos estáticos como árboles o edificios.
    c. **Física Básica**: Implementación de conceptos de física newtoniana, como posición, velocidad, aceleración y fuerza.
    d. **Manejo de Vectores**: Introducción y aplicación de vectores para manejar posiciones y movimientos.
    e. **Update vs Render**: Cálculo de posiciones y representación en pantalla.
    f. **Comportamientos de Persecución y Evasión**: Implementación de comportamientos como ‘perseguir’ y ‘escapar’.

5. **NPCs y Algoritmo Boids**
    a. **Implementación de NPCs**: Creación de personajes no jugables (NPCs) que interactúan con el protagonista.
    b. **Comportamientos Complejos**:
        i. **Zombies persiguiendo al protagonista**.
        ii. **Personajes que huyen de los zombies**.
        iii. **Personajes que persiguen al protagonista**.
    c. **Algoritmo Boids**:
        i. **Alineación**: Coordinación de la dirección de los NPCs.
        ii. **Separación**: Evitar colisiones entre NPCs.
        iii. **Cohesión**: Mantener a los NPCs agrupados.
    d. **Uso del zIndex**: Control de la superposición de los elementos gráficos en pantalla.

6. **Movimiento de Cámara + Interpolación Lineal**
    a. **Cálculo del Movimiento de Cámara**: Descripción del proceso para implementar un movimiento de cámara que sigue al protagonista del juego, moviendo todo el escenario de forma fluida.
    b. **Aplicación de Lerp al Movimiento de Cámara**: Implementación de interpolación lineal (Lerp) para suavizar el movimiento de la cámara
    c. **Explicación de Lerp**: Introducción al concepto de Lerp (Linear Interpolation), su funcionamiento y su aplicación en el contexto del desarrollo de videojuegos.
    d. **Aplicación de Lerp a la Velocidad del Personaje**: O a otras variables de física.
    e. **Interpolación Exponencial y Logarítmica**: Explicación de métodos avanzados de interpolación, como la exponencial y logarítmica, así como la función sigmoidea, para crear transiciones de movimiento más naturales.

7. **Grid y Tiles: Spatial Hashing**
    a. **Problemas con el Cálculo del Teorema de Pitágoras en Múltiples Objetos**: Análisis de las dificultades y limitaciones que surgen al calcular la distancia entre muchos objetos utilizando el teorema de Pitágoras, especialmente en términos de rendimiento computacional en juegos con numerosos elementos móviles.
    b. **Spatial Hashing: Funcionamiento Detallado**: Exploración en detalle del concepto de Spatial Hashing, una técnica que divide el espacio en celdas y asigna objetos a estas celdas para reducir el número de cálculos necesarios.

8. **Finite State Machine (FSM)**
    a. **Introducción a las Finite State Machines (FSM)**: Explicación de los conceptos básicos de una FSM y cómo se utilizan para gestionar comportamientos complejos en videojuegos.
    b. **Estados, acciones y sprites**
    c. **Cambio de Estado** según variables propias y del entorno de cada NPC, y toma de decisiones según estado.

9. **Sistema de Animación**
    a. **Entorno -> Estado -> Acción -> Selección de Sprite**: Según el estado proporcionado por la FSM, las propias variables del NPC y de su entorno, el sistema de animación selecciona y muestra la secuencia de imágenes o sprites adecuados para representar el comportamiento del NPC en ese estado.

10. **Colisiones y Motor de Física 2D**
    a. **Colisiones sin motor de física**.
    b. **Motores de física 2D**:
        i. Introducción a motores como Matter.js y Box2D.
        ii. Integración básica de MatterJS con PixiJS.
        iii. Ejemplos de uso: simulación de gravedad, rebotes, fricción, etc.

11. **Optimización**
    a. **Secuencia de frames vs Animación Esqueletal:** Spine, Adobe Animate
    b. **Quadtree**:
        i. Explicación del algoritmo Quadtree para optimización de colisiones.
        ii. Implementación de un Quadtree básico en JavaScript.
        iii. Ejemplos de cómo un Quadtree mejora el rendimiento en detección de colisiones.
    c. **Aproximación de Pitágoras**:
        i. Explicación del cálculo de distancias utilizando el teorema de Pitágoras.
        ii. Métodos de aproximación para evitar cálculos complejos.
        iii. Implementación de aproximaciones y su impacto en el rendimiento.
    d. **Distancias Precalculadas**:
        i. Concepto de precálculo y caching en el desarrollo de juegos.
        ii. Ejemplo de una lookup table para distancias entre celdas en una grid.

iii. Comparación de rendimiento entre cálculos en tiempo real y distancias precalculadas.

12. **Cordova, ElectronJS, Itch.io, PWAs.**
    a. **Publicación de juegos en diferentes plataformas**:
        i. Introducción a Cordova/Capacitor y ElectronJS para portar juegos a dispositivos móviles y escritorio.
        ii. Pasos básicos para empaquetar un juego con Cordova y ElectronJS.
        iii. Progressive Web Apps, pros y contras.
    b. **Marketing y estrategia de lanzamiento**:
        i. "Venderlo primero y desarrollarlo después": La importancia de validar ideas antes de invertir tiempo y recursos.
        ii. Proof of Concept: Cómo desarrollar una demo básica para obtener feedback temprano.
        iii. Plan de negocios para un juego independiente: cómo estructurar una propuesta de valor.
    c. **Reflexión sobre el desarrollo de software y Monetización**:
        i. Discusiones sobre diferentes modelos de negocios aplicados al desarrollo de videojuegos independientes; costo de adquisición publicitaria por usuario.