Acerca del

Videojuego

usando el

Motor Phaser

y el sistema de control de versiones

Git

y almacenado en

GitHub*

trabajo práctico para

Programación Orientada A Objetos

Profesor: Mariano D'agostino

F Emmanuel Messulam Franco I Vallejos Vigier

Instituto Politécnico Superior de Rosario "Gral. San Martin"

*https://github.com/EmmanuelMess/School-Phas3r-Game-Project

Introducción

Se presenta el trabajo practico, un juego donde el jugador tiene que luchar contra zombies para pasar el nivel. El nivel de dificultad es progresivo, es decir no se empieza con una oleada de 15. El nivel de dificultad crece exponencialmente, véase el array `EnemigosACrear` en el archivo "Principal.js".

A pesar de que el juego no posee tutorial, creemos intuitiva la forma de interaccion, (las teclas WASD para moverse, que son ampliamente usadas, y el espacio para disparar son una de las primeras cosas que, creemos, el jugador va a intentar).

Desafios

Desafíos no se nos presentaron en la forma en la que normalmente aparecen en proyectos con cientos de miles, o millones, de lineas. Si bien hubo que lidiar con el desconocimiento general de algunas herramientas proveídas por la plataforma GitHub, como la "Pull request", que no existe en git.

No se presento mayor dilema a la hora de separar el proyecto en archivos, ya que no iba a ser mantenido por un largo periodo de tiempo, se decidió poner todo en un solo archivo "Principal.js".

Tampoco se presento mayor dilema para separar todo en clases, ya que las ideas de "objeto" son muy fáciles de implementar en un videojuego, cosas como una 'bala' o un 'zombi' son claramente clases; y, claramente, el juego en sí es un objeto del cual solo existe una instancia, "singleton". Aunque esta ultima no esta expresada en código como tal, el juego es simplemente un diccionario de funciones, por la forma en que Phaser trata al juego y como ECMAScript trata a las clases y objetos.

Ejemplos

Instancia de clase

El juego crea cada enemigo por medio de:

```
let enemigo = new Enemy(JUEGO,
        JUEGO.rnd.integerInRange(xEnemigo-ESPARCIMIRENTO, xEnemigo+ESPARCIMIRENTO),
        165);
```

Donde `JUEGO` es la instancia de `Phaser.Game`; `JUEGO.rnd.integerInRange(xEnemigo-ESPARCIMIRENTO, xEnemigo` es la posicion en x de el enemigo, dado por `xEnemigo` que es el punto central donde se va a generar la oleada de enemigos y el `ESPARCIMIENTO` que es el radio de generación; y, por ultimo, `165` es la posición en y de el enemigo, que empieza desde la esquina superior izquierda como el 0 y aumenta positivamente hacia abajo.

Método publico

En verdad todos los metodos de clase son publicos en JavaScript, aca se tiene un metodo publico:

```
load() {
    this.animations.add('QuietMummy', [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
    this.body.collideWorldBounds = true;
}
```

Este hace que se cargue el enemigo, primero carga una animacion que se repite al terminar y está a a 7 FPS y luego le dice a motor fisico que debe el enemigo debe colisionar con el piso.

Método privado

Si bien existiría quien presentare una solución, los métodos privados de clase no existen en JavaScript. La solución en cuestión es crear una función anónima en el scope de un método publico:

```
(function () {}).prototype.f = function() {
    let funcionPrivada = function (instancia) {
    };
    funcionPrivada(this);//Uso
};
```

Cuando el scope de `f` termina `funcionPrivada` deja de existir, efectivamente haciendola privada. Pero se aprecian agujeros es esta idea al poco tiempo de usarla, quisierase usar de miembro deberasele pasar una instancia de la clase, aquí `instancia`, ademas, quisierasela usar en otros metodos publicos o privados deberasela compiar a todos, produciendo los problemas comunes con codigo duplicado (necesidad de modificar todas las copias para cambiar la funcion, aumento del espacio que ocupan de los archivos fuente, etc.).

Clase

La clase `Bala`, controla las balas.

```
class Bala extends Phaser.Sprite {
    constructor(juego, x, y) {
        super(juego, x, y, 'bala');
        Balas.push(this);
        this.creacionTiempo = Date.now();
        this.cargado = false;
    }
    load() {
        this.body.gravity = 0;
        if (!Marco.getIzquierdaODerecha()) {
            this.body.velocity.x = VELOCIDAD_BALAS;
            this.scale.setTo(1, 1);
        } else {
            this.body.velocity.x = -VELOCIDAD_BALAS;
            this.x = Marco.x - 30;
            this.scale.setTo(-1, 1);
        }
    }
    updateElem() {
        if (!this.cargado) {
            this.load();
            this.cargado = true;
        }
        if (this.x > 800) {
            let indice = Balas.indexOf(this);
            if (indice > -1) {
                Balas.remove(indice);
            }
            this.kill();
        }
    }
}
```

Atributo de clase

Este es un atributo de la clase `ControladorFondo`, sirve para saber cuanto del fondo se movio hacia la izquierda, para poder dar posiciones absolutas del mundo:

```
this.posicionAbsolutaDeInicioDeMundo = 0;
```

Constructor

Aca se tiene un constructor (de `MarcoPlayer`):

```
constructor(juego, x, y) {
    super(juego, x, y, 'Quieto');
    this.enfriadoBalasTiempoInicio = 0;
    this.balasEnCargador = BALAS_EN_CARGADOR;
        this.izquierdaODerecha = false;
    this.animacion = AnimacionEnum.quieto;
}
```

Se le pasa un juego y ambas coordenadas para posicionarlo. Llama a el constructor de la superclase `Phaser.Sprite` en este caso. luego determina el valor de algunas variables de clase que luego se usaran. Puede apreciarse el uso de un enum en `this.animacion = AnimacionEnum.quieto;`.

Mejoras posibles

Para una guía más actualizada y más completa véase la pagina de issues (filtrando por "enhancements") en GitHub[1].

Un sistema de niveles

Más niveles, un menú con más niveles, y conforme se avance se desbloqueé nuevos personajes. Un sistema de progreso donde el jugador pueda avanzar y enfrentarse a enemigos más fuertes. Al implementar un sistema de logros, buscamos que el usuario vuelva a jugar niveles anteriores, por el simple hecho de adquirir contenido desbloqueable exclusivo (armas por ejemplo), el cual no sería posible de comprar o conseguir en la tienda.

Un mecanismo de experiencia

Un sistema de experiencia con contenido desbloqueable para otorgar más fuerza en los ataques, creación de un HUD. Mejoras graduales al jugador ayudarian a contrarrestar los enemigos más fuertes si se implementaran más niveles. Implementar un BOSS al final de cada nivel.

Un sistema que permita un juego cooperativo para el usuario, así como la posibilidad de escoger personajes, distintos, al comienzo de cada nuevo nivel. Agregar dinámica al juego con consumibles, misiones secundarias, medallas o logros, además de vehículos. Nuestra idea en el desarrollo, fue realizar un juego en el cual se incremente rápidamente la dificultad del mismo, por eso sería ideal implementar un mecanismo de guardado o checkpoint.

Implementar plataformas, con las cuales el usuario pueda interactuar y de ahí poder sacar ventaja de sus enemigos.

Actualizar a la versión 3 del motor Phaser

Como dice el FAQ (preguntas frecuentes, por sus siglas en ingles) de Phaser 3 (traducción)[2]:

> ...como v3 otorga un montón de nuevas formas de lograr objetivos comunes. Te darán la posibilidad de que tu código sea más claro y más estructurado. Aprovechalo.

[1] `https://github.com/EmmanuelMess/School-Phas3r-Game-Project/issues?q=is%3Aissue+is%3Aopen+sort%3Aupdated-desc+label%3Aenhancement`

[2] Véase la pregunta tres en `https://phaser.io/phaser3/faq`

Pero aún hay pocos ejemplos y documentación con respecto a la nueva versión, por lo que se prefirió usar la v2. En un futuro debería moverse el proyecto para aprovechar la nueva versión.