Anabel Mena Meseguer y Elena Vallejo Ruiz

SISTEMAS GRÁFICOS
PRÁCTICA 2:
DISEÑO E IMPLEMENTACIÓN DE UN JUEGO BASADO EN “BATTLE CHESS”

UNIVERSIDAD
DE GRANADA

ÍNDICE

1. DESCRIPCIÓN DEL JUEGO

A la largo de nuestra segunda práctica de sistemas gráficos hemos decidido desarrollar un Battle Chess llamado MysticChess, el cual sigue una temática de fantasía donde caballeros armados con espadas y escudos combaten contra magos armados con varitas.

La batalla/animación principal en nuestro juego es llevada a cabo por todas nuestras piezas, en ella, cuando una ficha se coma a otra ficha cualquiera que no sea una torre se iniciará la animación de captura, la cual consistirá en que la pieza, la cual tiene brazos articulados con un escudo y una espada o una varita, hará un movimiento de ataque en el que mueve la arma de forma horizontal cortando la cabeza de esta pieza, la cual caerá al suelo, posteriormente será enviada al "Cementerio" que es la zona donde se acumulan las piezas comidas por los respectivos equipos.

Todos los modelos utilizados para las piezas los hemos diseñado nosotras mismas mediante Three.js y para ir desarrollando la aplicación hemos utilizado un repositorio GitHub https://github.com/anabelmenam/SG_UGR.

Para iniciar el menú basta con en una terminal donde se encuentre el servidor ejecutar: 'python3 server-launcher.py' y se encontrarán las diferentes opciones, como puede ser ver la animación de combate, jugar o ver el tablero. También se puede escoger entre las diferentes piezas para visualizar.

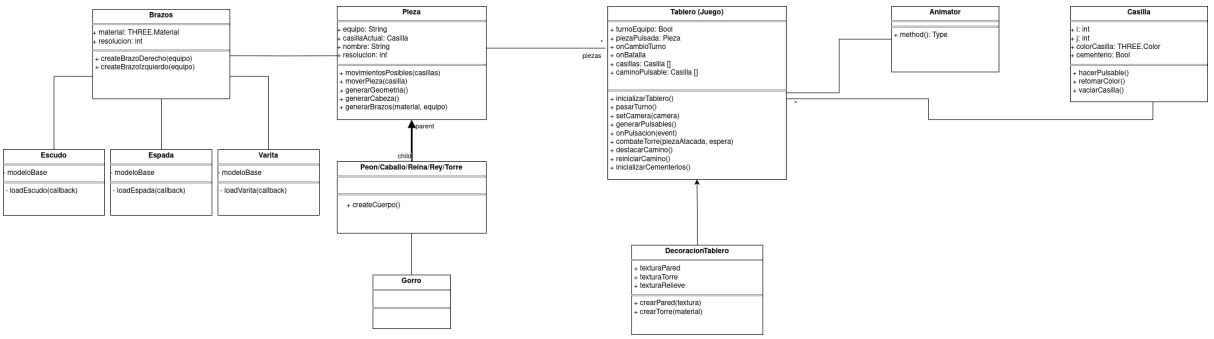
2. DISEÑO DE LA APLICACIÓN

El diseño de la aplicación gira en torno a una clase principal llamada Tablero (Juego), la cual funciona como controlador del juego e interactúa con la escena (MyScene) enviando todos los modelos (Piezas, tablero, decoracion) que se muestra en pantalla, además de controlar el cambio de turnos, lo cual nos permite girar la cámara, modificar la iluminación…

Algunas animaciones se realizan con la clase Animator, que tiene una función setAndStart con el que podemos indicar la posición inicial y final de la animación,

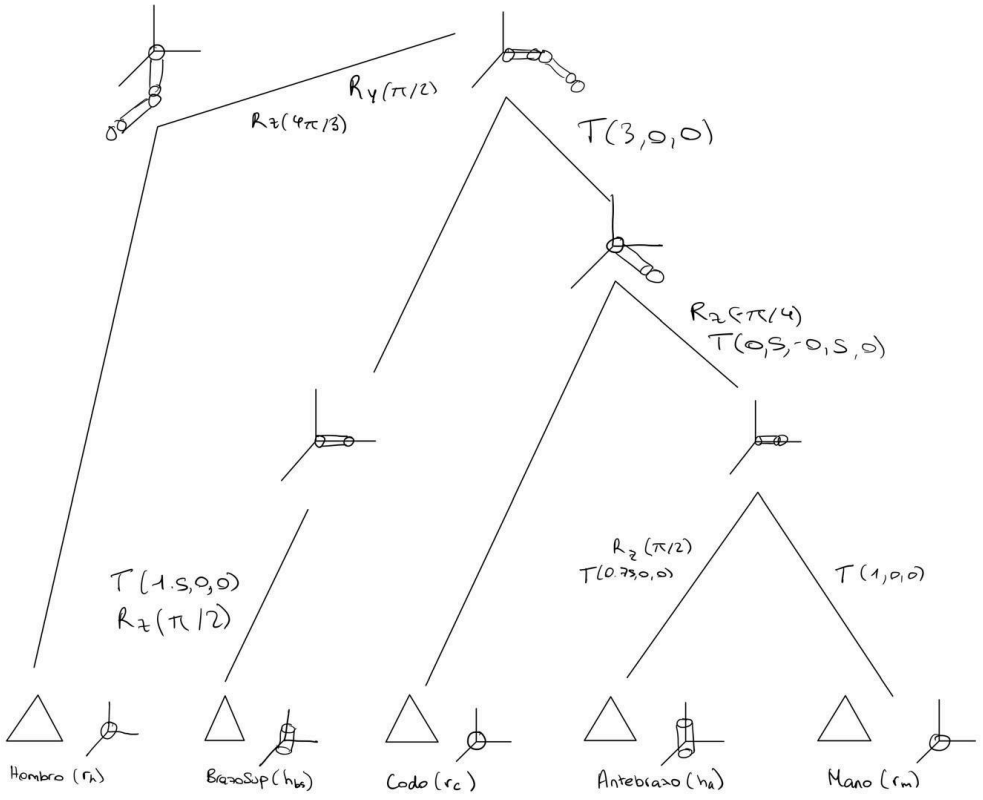
además de cuanto va a durar, mientras que otras hemos optado por hacerlas directamente en la clase que se fuesen a utilizar.

2.1. DIAGRAMAS DE CLASE



2.2. MODELOS JERÁRQUICOS

Nuestro modelo jerárquico es el brazo de las piezas



2.3. ALGORITMOS DE ALTA IMPORTANCIA

Tenemos 2 algoritmos de alta importancia, la animación de combate y el pick.

El pick: **OnPulsación(evento)** se trata de la función encargada de administrar las pulsaciones de nuestro usuario, ya sea pulsando una pieza del equipo actual o pulsando la casilla donde esta pieza tiene que acabar. Desde MyScene se llama con un EventListener de click

Se empieza tomando las coordenadas del ratón y a partir de ellas y la cámara se genera el raycaster, una vez tenemos el raycaster se obtiene las piezas pulsadas a partir de aquellas que puedan ser pulsadas (las del equipo), estas han sido inicializadas previamente con la función: generarPulsables( ).

Nos quedamos con la primera pieza pulsable (piezasPulsadas[0]) y la almacenamos en piezaActual.

Luego entramos en un else if que se encarga de la lógica de selección y deselección,Si se hace click en la misma pieza ya seleccionada, se deselecciona: se baja su posición y se limpia la selección. Si se hace clic en una pieza diferente, se selecciona esa nueva pieza: Se reinicia el camino anterior, elevamos, y calculamos los movimientosPosibles…

Posteriormente entramos en la selección de una casilla (si hay una pieza seleccionada), volvemos a utilizar el raycaster pero esta vez con el camino viable de la pieza (esto ya depende de la lógica interna de la pieza según el tipo que sea) si la casilla pulsada no es la actual de la pieza: Si hay una pieza enemiga, se inicia un combate y si no hay pieza, la pieza se moverá directamente a esa casilla.

Tras terminar el combate se anima la pieza a su nueva posición y se reinician las variables necesarias, se cambia de turno, se regeneran las piezas pulsables porque se ha cambiado de equipo.

Código fuente:

```
onPulsacion(event) {
```

```
        this.mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
        this.mouse.y = 1 - 2 * (event.clientY / window.innerHeight);
        this.raycaster.setFromCamera(this.mouse, this.camera);

        // Intentamos primero pulsar una pieza del equipo actual
        const piezasPulsadas = this.raycaster.intersectObjects(this.piezasPulsables, true);
        if (piezasPulsadas.length > 0) {
            const intersect = piezasPulsadas[0];
            const piezaActual = intersect.object.userData.pieza;

            if (this.piezaPulsada && piezaActual === this.piezaSeleccionada) {
                // Deselección de la misma pieza
                this.reiniciarCamino();
                this.piezaSeleccionada.position.y = 0.5;
                this.piezaSeleccionada = null;
                this.piezaPulsada = false;
            } else {
                // Selección de nueva pieza
                this.reiniciarCamino();
                if (this.piezaSeleccionada) this.piezaSeleccionada.position.y = 0.5;
                this.piezaSeleccionada = piezaActual;
                this.piezaSeleccionada.position.y = 1;
                this.piezaPulsada = true;

                this.caminoPulsable = this.piezaSeleccionada.movimientosPosibles(this.casillas);
                this.destacarCamino();
            }

            return;
        }

        // Si no pulsamos una pieza, puede ser una casilla valida del camino
        if (this.piezaPulsada && this.caminoPulsable.length > 0) {
            const casillasPulsadas = this.raycaster.intersectObjects(this.caminoPulsable, true);
            if (casillasPulsadas.length > 0) {
                const casilla = casillasPulsadas[0].object.userData.casilla;
                if (casilla && casilla !== this.piezaSeleccionada.casilla) {
                    //Si la casilla tiene una pieza, la comemos
                    var espera = $.Deferred();
                    if(casilla.pieza != null) {

                        if(this.piezaSeleccionada instanceof Torre) {
                            this.combateTorre(casilla.pieza, espera);
                        }
                        else {
                            this.comerPieza(casilla.pieza, espera);
                        }
                    } else {
                        espera.resolve();
                    }
                    espera.done(() => {
                        var wait = $.Deferred();
                        const from = this.piezaSeleccionada.position;
                        const to = new THREE.Vector3(casilla.posI, 0.5, casilla.posJ);

                        this.animator.setAndStart(from, to, 500, wait);
                        wait.done(() => {
                            this.piezaSeleccionada.moverPieza(casilla);
                            this.piezaSeleccionada.position.y = 0.5;
                            this.reiniciarCamino();
```

```
                        this.piezaSeleccionada = null;
                        this.piezaPulsada = false;
                        this.pasarTurno();
                        this.generarPulsables();
                    })
                })

            }
        }
    }
}
```

La animación de combate: es llevada a cabo por la función combate(piezaAtacada, espera) , podemos dividirla en 7 fragmentos, inicialmente vamos a almacenar las posiciones iniciales de la pieza, para luego poder restaurarlas al final de la animación, lo primero que se hace es mover la pieza atacante hacia la atacada - 0.3 en z, luego movemos la pieza atacada en z + 0.3, esto para que compartan la casilla, luego se sube el brazo de la pieza atacante rotando en z 90 grados para hacer como si se estuviera cortando la cabeza, para finalizar movemos la cabeza para que gire y caiga al suelo y restauramos las posiciones de la pieza atacante, encadenamos todos los movimientos e iniciamos.

Código fuente:

```
combateTorre(piezaAtacada, espera) {
    const torre = this.piezaSeleccionada;
    const brazoDch = torre.brazoDch;
    const codo = brazoDch.codo;
    const hombro = brazoDch.hombro;
    const antebrazo = brazoDch.antebrazo;
    const mano = brazoDch.mano;


    const cabeza = piezaAtacada.cabeza;
    const rotacionCabeza = 0
    const rotacionIniAntebrazo  = antebrazo.rotation.z;
    const rotacionHombroY = hombro.rotation.y;
    const rotacionHombroZ = hombro.rotation.z;
    const posTorre = torre.position;
    const posAtacada = piezaAtacada.position;

    // 1. Mover pieza atacante
    let movimiento = 0;
    if(this.turnoEquipo === 0) {
        movimiento = +0.3;
    } else {
        movimiento = -0.3;
    }
    const moverPiezaAtacante = new TWEEN.Tween({ x: posTorre.x, z: posTorre.z , y: posTorre.y})
        .to({ x: posAtacada.x, z: posAtacada.z + (-1)*movimiento, y: posAtacada.y }, 500)
        .onUpdate(({ x, y, z }) => {
            torre.position.x = x;
            torre.position.z = z;
```

```
                torre.position.y = y;

            });

        // 2. Mover pieza atacada
        const moverPiezaAtacada = new TWEEN.Tween({  z: posAtacada.z })
            .to({ z: posAtacada.z + movimiento }, 500)
            .onUpdate(({ z }) => {
                piezaAtacada.position.z = z;
            });


        // 3. Subir brazo
        const subirBrazo = new TWEEN.Tween({ z: THREE.MathUtils.degToRad(rotacionHombroZ) })
            .to({ z: THREE.MathUtils.degToRad(90) }, 600)
            .onUpdate(({ z }) => {
                hombro.rotation.z = z;
        });

        // 4. Rotar brazo
        const rotarBrazo = new TWEEN.Tween({ x: rotacionIniAntebrazo, y: rotacionHombroY })
            .to({ x: THREE.MathUtils.degToRad(90),  z: THREE.MathUtils.degToRad(170)  }, 600)
            .onUpdate(({ x, y }) => {
                antebrazo.rotation.z = x;
                hombro.rotation.y = y;
        });

        // 5. Rotar la cabeza
        const rotarCabeza = new TWEEN.Tween({ z: rotacionCabeza })
            .to({ z: THREE.MathUtils.degToRad(90) }, 500)
                .onUpdate(({ z }) => {
                cabeza.rotation.z = z;
        });

        // 6. Caer la cabeza
        const caerCabeza = new TWEEN.Tween({ x: cabeza.position.x, y: cabeza.position.y })
            .to({ x: cabeza.position.x, y: 0.5 }, 900)
            .onUpdate(({ x, y }) => {
                cabeza.position.x = x;
                cabeza.position.y = y;
        });


        // 7. Volver a la posición inicial
        const volverOrigen = new TWEEN.Tween({ x: hombro.rotation.z, z: antebrazo.rotation.z, y:
hombro.rotation.y })
            .to({ x: rotacionHombroZ, z: rotacionIniAntebrazo, y: rotacionHombroY  }, 600)
            .onUpdate(({ x, y, z }) => {
                hombro.rotation.z = x;
                hombro.rotation.y = y;
                      antebrazo.rotation.z = z;
        }).onComplete(() => {
                this.comerPieza(piezaAtacada, espera);
                //let cementerio = piezaAtacada.equipo === 0 ? this.cementerioBlancas.pop() :
this.cementerioNegras.pop()
                //cementerio.setPieza(piezaAtacada);
            });

        moverPiezaAtacante.chain(moverPiezaAtacada);
```

```
        moverPiezaAtacada.chain(subirBrazo);
        subirBrazo.chain(rotarBrazo);
        rotarBrazo.chain(rotarCabeza);
        rotarCabeza.chain(caerCabeza);
        caerCabeza.chain(volverOrigen);
        moverPiezaAtacante.start();
    }
```

3. MODELOS IMPORTADOS

Para buscar los modelos, hemos utilizado principalmente la página de cults3d, que tiene una gran variedad de modelos 3D los cuales se pueden llegar a filtrar hasta por tipo de archivo/extensión:

- Varita: https://cults3d.com/:440445
- Espada: https://cults3d.com/:788584
- Escudo: https://cults3d.com/:2543199

4. CONCLUSIÓN

Estas prácticas nos han resultado muy gratificantes e interesantes, ya que nunca antes habíamos tenido la oportunidad de desarrollar nuestro propio juego desde cero durante la carrera. Poder hacerlo ha sido una experiencia enriquecedora tanto a nivel técnico como creativo. Además, trabajar con Three.js ha sido especialmente positivo: se trata de una biblioteca muy potente y accesible, que facilita enormemente el desarrollo de gráficos 3D en comparación con lo que vimos el año pasado en la asignatura de Informática Gráfica. La mejora es evidente, tanto en términos de facilidad de programación como en los resultados visuales que se pueden conseguir.