**Trabajo Final de Equivalencia de FPOO**

**Documentación Técnica**

Nombre: Lautaro Eduardo

Apellido: Espinosa

DNI: 44349863

Carrera: Tecnicatura en diseño integral de videojuegos

Año: 2025

Historia de usuario

|  |  |
| --- | --- |
| Código: 001 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: construcción de escenario y ubicación de game objects | |
| Prioridad: Alta | Riesgo de desarrollo: Alto |
| Estimación: 3 dias | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Como jugador quiero observar un escenario abierto con enemigos y cofres de la suerte | |
| Criterios de aceptación:  CA1: El jugador puede moverse en 8 direcciones mediante la clase JoyPad  CA2: Se crea una clase enemigo y se generan una cantidad de enemigos fija repartida aleatoriamente por el mapa  CA3: Los enemigos en pantalla pueden o no moverse por el escenario pero respetan los limites del mapa  CA4: Se generan dentro del lienzo del mapa (2400x1600) una cantidad finita esparcida aleatoriamente de cofres de la suerte    Movimiento del Protagonista Movimiento del enemigo    generación de cofres de la suerte | |
| Observaciones:  CA1: El protagonista puede moverse hasta los limites establecidos del mapa (2400x1600)  CA3: Para mantener la consistencia del juego en diferentes equipos se utilizará DeltaTime | |

Tabla EPS ( Entrada – Proceso – Salida)

|  |  |
| --- | --- |
| Entrada | Parámetros del mapa (2400x1600), clases Protagonista, Enemigo, Lucky, JoyPad |
| Proceso | Se crean objetos del protagonista, enemigos y cofres de la suerte con posiciones aleatorias; se limita el movimiento del protagonista y enemigos al mapa. |
| Salida | Escenario inicial con el protagonista controlable, enemigos y cofres distribuidos aleatoriamente. |
|  |  |

CA1:

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.

CA2 y CA3:

Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

CA4:

Imagen de la pantalla de una ventana

El contenido generado por IA puede ser incorrecto.

**private** Protagonista prota; /\*\* Atributo de la clase del protagonista\*/

**private** JoyPad joypad; /\*\* Atributo de l clase para movimiento \*/

**private** ArrayList<Lucky> luckyblock; /\*\* Atributo para generar los luckyblocks\*/

**private** Protagonista prota; /\*\* Atributo de la clase del protagonista\*/

**private** JoyPad joypad; /\*\* Atributo de l clase para movimiento \*/

**private** ArrayList<Lucky> luckyblock; /\*\* Atributo para generar los luckyblocks\*/

frameRate(**60**); /\*\*Ajustar a 60 FPS \*/ /\*\* \*/

prota = **new** Protagonista(); /\*\* Inicializar protagonista\*/

prota.setPosicion(**new** PVector(width / **2**, height / **2**)); /\*\* Establecer posicion en el centro de la pantalla\*/

prota.setVelocidad(**new** PVector(**5**, **5**)); /\*\*Establecer la velocidad base\*/

joypad = **new** JoyPad(); /\*\* Clase joypad para movimiento del protagonista \*/

luckyblock = **new** ArrayList<Lucky>(); /\*\* Se crea un arraylist para la generacion de multiples cofres de la suerte \*/

/\*\* Generacion de Luckyblocks de manera aleatoria en una cantidad de 15 esparcidos por el mapa\*/

**for** (**int** j = **0**; j < **15**; j++) {

PVector posicionL = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

**int** tipo = (**int**) random(**2**);

**float** duracion = random(**0**,**15**); /\*\* Duracion del buffo o penalizacion\*/

**if** (tipo == **0**){ /\*\* Tipo de suerte\*/

luckyblock.add(**new** LuckyBonus(posicionL,(**int**) random(**0**,**200**),duracion));

}

**else** **if** (tipo == **1**){

luckyblock.add(**new** LuckyPenalty(posicionL,(**int**) random(**0**,**150**),duracion));

}

luckyblock.add(**new** Lucky(posicionL, lucky));

}

/\*\* Se inicializa una Lista de 8 enemigos cuya posicion será aleatoria en un espacio de 2400x1600 \*/

enemigos = **new** ArrayList<Enemigo>(); /\*\* Se crea un arraylist para la generacion de multiples enemigos de la suerte \*/

**for** (**int** i = **0**; i < **8**; i++) {

PVector posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigos.add(**new** EnemigoFuerte(posicion,enemigo2));

posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigos.add(**new** EnemigoLento(posicion,enemigo1)); //corregir

posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigos.add(**new** EnemigoRapido(posicion,enemigo3));

}

**class** **Protagonista** {

**private** PVector posicion; /\*\* Atributo para la posicon\*/

**private** PVector velocidad; /\*\* Atributo para la velocidad\*/

**private** **float** radio = **20**; /\*\* Hitbox del protagonista\*/

**public** **Protagonista**() { /\*\* metodos accesores\*/

**this**.posicion = **new** PVector(width/**2**, height/**2**);

**this**.velocidad = **new** PVector(**5**, **5**);

**this**.protagonista = protagonista;

}

/\*\* \*/

/\*\* funcion para dibujar al protagonista + dibujo del proyectil al invocarse\*/

**public** **void** **display**(PImage protagonista){

fill(**255**,**0**,**0**);

strokeWeight(**5**);

circle(**this**.posicion.x,**this**.posicion.y,**40**);

**for** (Proyectil p : proyectiles){

p.display();

}

}

/\*\* logica del moviimiento y colisiones con el borde del mapa \*/

**public** **void** **mover**(**int** direccion, **float** deltaTime) {

**float** velocidadActual = **300** \* deltaTime;

**switch**(direccion) {

**case** **0**: /\*\*Arriba \*/

**this**.posicion.y -= velocidadActual;

**if** (**this**.posicion.y < **0**) **this**.posicion.y = **0**;

**break**;

**case** **1**: /\*\* Derecha\*/

**this**.posicion.x += velocidadActual;

**if** (**this**.posicion.x > **2400**) **this**.posicion.x = **2400**;

**break**;

**case** **2**: /\*\*Abajo \*/

**this**.posicion.y += velocidadActual;

**if** (**this**.posicion.y > **1600**) **this**.posicion.y = **1600**;

**break**;

**case** **3**: /\*\* Izquierda\*/

**this**.posicion.x -= velocidadActual;

**if** (**this**.posicion.x < **0**) **this**.posicion.x = **0**;

**break**;

}

}

**public** **void** **setPosicion**(PVector posicion){

**this**.posicion= posicion;

}

**public** **void** **setVelocidad**(PVector velocidad){

**this**.velocidad= velocidad;

}

**class** **Enemigo** {

/\*\* Atributos de la clase enemigo, tipo protected para poder ser modificados segun el tipo de enemigo que se herede \*/

**protected** PVector posicion; /\*\* \*/

**protected** PVector velocidad; /\*\* \*/

**protected** **float** radio = **20**; /\*\* \*/

**protected** color colorEnemigo; /\*\* \*/

/\*\* \*/

**public** **Enemigo**(PVector posicion) {

**this**.posicion = posicion;

**this**.velocidad = **new** PVector(random(**60**), random(**60**)); //modificar velocidad

**this**.radio = **20**;

**this**.colorEnemigo = color(**255**,**0**,**0**);

}

/\*\* funcion para dibujar al enemigo base\*/

**public** **void** **display**(){

fill(colorEnemigo);

strokeWeight(**5**);

**for** (Proyectil p : proyectiles) {

p.display();

}

}

/\*\* funcion para actualizar a cada enemigo del campo en movimiento y disparo\*/

**public** **void** **update** (Protagonista prota, **float** deltaTime){

mover(deltaTime);

}

/\*\* logica del moviimiento y colisiones con el borde del mapa \*/

**public** **void** **mover**(**float** deltaTime) {

**this**.posicion.add(PVector.mult(**this**.velocidad, deltaTime)); /\*\* logica del moviimiento y colisiones con el borde del mapa Ajustado por deltaTime \*/

/\*\* verifica si el enemigo se encuentra en los bordes y lo hace rebotar como el logo de dvd\*/

**if** (**this**.posicion.x < **0**) {

**this**.posicion.x = **0**;

**this**.velocidad.x \*= -**1**;

}

**if** (**this**.posicion.x > **2400**) {

**this**.posicion.x = **2400**;

**this**.velocidad.x \*= -**1**;

}

**if** (**this**.posicion.y < **0**) {

**this**.posicion.y = **0**;

**this**.velocidad.y \*= -**1**;

}

**if** (**this**.posicion.y > **1600**) {

**this**.posicion.y = **1600**;

**this**.velocidad.y \*= -**1**;

}

}

}

**public** PVector **getPosicion**() {

**return** **this**.posicion;

}

**public** **float** **getRadio**() {

**return** **this**.radio;

}

}

frameRate(**60**); /\*\*Ajustar a 60 FPS \*/ /\*\* \*/

prota = **new** Protagonista(); /\*\* Inicializar protagonista\*/

prota.setPosicion(**new** PVector(width / **2**, height / **2**)); /\*\* Establecer posicion en el centro de la pantalla\*/

prota.setVelocidad(**new** PVector(**5**, **5**)); /\*\*Establecer la velocidad base\*/

joypad = **new** JoyPad(); /\*\* Clase joypad para movimiento del protagonista \*/

luckyblock = **new** ArrayList<Lucky>(); /\*\* Se crea un arraylist para la generacion de multiples cofres de la suerte \*/

/\*\* Generacion de Luckyblocks de manera aleatoria en una cantidad de 15 esparcidos por el mapa\*/

**for** (**int** j = **0**; j < **15**; j++) {

PVector posicionL = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

**int** tipo = (**int**) random(**2**);

**float** duracion = random(**0**,**15**); /\*\* Duracion del buffo o penalizacion\*/

**if** (tipo == **0**){ /\*\* Tipo de suerte\*/

luckyblock.add(**new** LuckyBonus(posicionL,(**int**) random(**0**,**200**),duracion));

}

**else** **if** (tipo == **1**){

luckyblock.add(**new** LuckyPenalty(posicionL,(**int**) random(**0**,**150**),duracion));

}

luckyblock.add(**new** Lucky(posicionL, lucky));

}

/\*\* Se inicializa una Lista de 8 enemigos cuya posicion será aleatoria en un espacio de 2400x1600 \*/

enemigos = **new** ArrayList<Enemigo>(); /\*\* Se crea un arraylist para la generacion de multiples enemigos de la suerte \*/

**for** (**int** i = **0**; i < **8**; i++) {

PVector posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigos.add(**new** EnemigoFuerte(posicion,enemigo2));

posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigos.add(**new** EnemigoLento(posicion,enemigo1)); //corregir

posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigos.add(**new** EnemigoRapido(posicion,enemigo3));

}

/\*\*keyPressed para interaccion del input con el videojuego \*/

**public** **void** **keyPressed**() {

**if** (estadoActual == ESTADO\_JUGANDO || estadoActual == ESTADO\_NIVEL2 || estadoActual == ESTADO\_HORDA){ /\*\* El protagonista se movera siempre y cuando se encuentre en algunos de los estados de juego\*/

**if** (key == 'w' || keyCode == UP) { /\*\* mover hacia arriba con flecha o w\*/

joypad.setUpPressed(**true**);

}

**if** (key == 's' || keyCode == DOWN) { /\*\* mover hacia arriba con flecha o s\*/

joypad.setDownPressed(**true**);

}

**if** (key == 'd' || keyCode == RIGHT) { /\*\* mover hacia arriba con flecha o d\*/

joypad.setRightPressed(**true**);

}

**if** (key == 'a' || keyCode == LEFT) { /\*\* mover hacia arriba con flecha o a\*/

joypad.setLeftPressed(**true**);

}

**if** (key == ' ') { /\*\* Ejecutar disparo con la tecla espacio\*/

prota.disparar();

}

}

}

**if** (key == ESC) { /\*\* cerrar aplicacion \*/

exit();

}

}

/\*\* control de teclas cuando dejan de ser pulsadas \*/

**public** **void** **keyReleased**() {

**if** (key == 'w' || keyCode == UP) {

joypad.setUpPressed(**false**);

}

**if** (key == 's' || keyCode == DOWN) {

joypad.setDownPressed(**false**);

}

**if** (key == 'd' || keyCode == RIGHT) {

joypad.setRightPressed(**false**);

}

**if** (key == 'a' || keyCode == LEFT) {

joypad.setLeftPressed(**false**);

}

}

**class** **JoyPad**{

**private** **boolean** upPressed;

**private** **boolean** downPressed;

**private** **boolean** rightPressed;

**private** **boolean** leftPressed;

**public** **boolean** **isUpPressed**(){

**return** **this**.upPressed;

}

**public** **void** **setUpPressed**(**boolean** upPressed){

**this**.upPressed = upPressed;

}

**public** **boolean** **isDownPressed**(){

**return** **this**.downPressed;

}

**public** **void** **setDownPressed**(**boolean** downPressed){

**this**.downPressed = downPressed;

}

**public** **boolean** **isRightPressed**(){

**return** **this**.rightPressed;

}

**public** **void** **setRightPressed**(**boolean** rightPressed){

**this**.rightPressed = rightPressed;

}

**public** **boolean** **isLeftPressed**(){

**return** **this**.leftPressed;

}

**public** **void** **setLeftPressed**(**boolean** leftPressed){

**this**.leftPressed = leftPressed;

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 002 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: construcción de una maquina de estado para el manejo del juego | |
| Prioridad: Alta | Riesgo de desarrollo: Alto |
| Estimación: 3 dias | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Una maquina de estado que permitirá moverse en distintas instancias del videojuego | |
| Criterios de aceptación:  CA1: Se debe crear el estado menu  CA2: Se debe crear el estado nivel 1  CA3: Se debe crear el estado Game Over  CA4: Se debe crear el estado pausa    menú Principal menú de pausa    Pantalla de gameOver | |
| Observaciones:  El estado menú,gameover,pausa se dibujaran en una clase aparte llamada GestorPantallas  El atributo estadoG es un estado temporal que almacena el estado de la maquina al entrar y salir de la pausa | |
| Tabla EPS ( Entrada – Proceso – Salida)   |  |  | | --- | --- | | Entrada | Acciones del jugador (clics y teclas), estados definidos (MENU, JUGANDO, PAUSA, GAMEOVER) | | Proceso | Se gestiona el flujo entre pantallas del juego mediante una máquina de estados y la clase GestorPantallas. | | Salida | Cambios entre pantallas: menú, juego, pausa y game over según las acciones del jugador. | | |

Diagrama

El contenido generado por IA puede ser incorrecto.

/\*\* Estados del videojuego\*/

**final** **int** ESTADO\_MENU = **0**;

**final** **int** ESTADO\_JUGANDO = **1**;

**final** **int** ESTADO\_GAMEOVER = **2**;

**final** **int** ESTADO\_NIVEL2MENU = **7**;

**private** **int** estadoActual = ESTADO\_MENU;

**final** **int** ESTADO\_PAUSA = **3**;

**final** **int** ESTADO\_NIVEL2 = **4**;

**final** **int** ESTADO\_CREDITOS = **5**;

**final** **int** ESTADO\_HORDA= **6**;

**int** estadoG;

/\*\* Maquina de estado que llama a diferentes funciones segun el estado actual\*/

**switch**(estadoActual){

**case** ESTADO\_MENU:

gestorPantallas.dibujarMenu(); /\*\* Llama a la funcion de dibujar el menu al abrir la aplicacion para poder interactuar con las opciones del juego \*/

**break**;

**case** ESTADO\_JUGANDO:

actualizarJuego(deltaTime); /\*\* Es el estado referido para el primer nivel del videojuego \*/

**break**;

**case** ESTADO\_GAMEOVER:

gestorPantallas.mostrarGameOver(); /\*\* Es la pantalla que se mostrara al perder la cantidad de vidas totales del protagonista\*/

**break**;

**case** ESTADO\_PAUSA:

gestorPantallas.mostrarPausa(); /\*\* Es el estado de pausa que se puede acceder en cualquier momento mediante la tecla P\*/

**break**;

}

/\*\* MousePressed para interactuar en el videojuego con el mouse \*/ /\*\* \*/

**void** **mousePressed**() {

/\*\* ====== BOTÓN INICIAR JUEGO ====== \*/

**if** (mouseX > width / **2** - **150** && mouseX < width / **2** + **150** &&

mouseY > **500** && mouseY < **570**) {

estadoActual = ESTADO\_JUGANDO;

}

/\*\* ====== BOTÓN Salir ====== \*/

**else** **if** (mouseX > width / **2** - **150** && mouseX < width / **2** + **150** &&

mouseY > **700** && mouseY < **770**) {

exit();

}

}

/\*\*keyPressed para interaccion del input con el videojuego \*/

**public** **void** **keyPressed**() {

**if** (estadoActual == ESTADO\_JUGANDO || estadoActual == ESTADO\_NIVEL2 || estadoActual == ESTADO\_HORDA){ /\*\* El protagonista se movera siempre y cuando se encuentre en algunos de los estados de juego\*/

**if** (key == 'p' || key == 'P') { /\*\* cambiar el estado de juego a pausa siempre y cuando se encuentre en algunos de los estados de juego (no incluye menus obviamente)\*/

**if** (estadoActual == ESTADO\_JUGANDO || estadoActual == ESTADO\_NIVEL2 || estadoActual == ESTADO\_HORDA ) {

estadoG = estadoActual; /\*\* Guarda el estado actual entre las 3 posibilidades \*/

estadoActual = ESTADO\_PAUSA;

} **else** **if** (estadoActual == ESTADO\_PAUSA) { /\*\* en caso de estar en el estado de pausa se reanuda el juego\*/

estadoActual = estadoG; /\*\* Vuelve al estado anterior \*/

}

}

**if** (key == ESC) { /\*\* cerrar aplicacion \*/

exit();

}

**if** (key == 'r' && estadoActual == ESTADO\_GAMEOVER) { /\*\* En el caso que el estado se haya cambiado a GAMEOVER el jugador presionando r puede volver al estado jugando desde el primer nivel\*/

reiniciarJuego(); /\*\* reinicia el juego en su totalidad antes de cambiar el estado\*/

estadoActual = ESTADO\_JUGANDO;

}

**if** (key == 'e' && estadoActual == ESTADO\_GAMEOVER) { /\*\* En otro caso el jugador puede optar por ingresar al menu princial desde el gameover\*/

reiniciarJuego(); /\*\* reinicia el juego en su totalidad antes de cambiar el estado\*/

estadoActual = ESTADO\_MENU;

}

**if** (key == 'r' && estadoActual == ESTADO\_GAMEOVER) { /\*\* por alguna razon esta duplicado, ver \*/

audio.detenerMusicaActual(); // detener cualquier música activa

reiniciarJuego();

estadoActual = ESTADO\_JUGANDO;

}

**if** (key == 'e' && estadoActual == ESTADO\_GAMEOVER) { /\*\* por alguna razon esta duplicado, ver \*/

audio.detenerMusicaActual();

reiniciarJuego();

estadoActual = ESTADO\_MENU; // volver al menu desde el gameover

}

**if** (key == 'r' && estadoActual == ESTADO\_CREDITOS) { /\*\* Estado creditos del videojuego, se regresa al menu mediante la r \*/

audio.detenerMusicaActual();

reiniciarJuego();

estadoActual = ESTADO\_MENU;

}

}

**class** **GestorPantallas** {

PFont fontTitulo, fontTexto; /\*\* Fuentes del gestor de pantallas\*/

ArrayList<Integer> mejoresPuntuaciones;

**float** volumen;

**boolean** musicaActiva;

Protagonista prota; /\*\* objeto protagonista\*/

GestorPantallas(PFont fontTitulo, PFont fontTexto, ArrayList<Integer> mejoresPuntuaciones,

**float** volumen, **boolean** musicaActiva, Protagonista prota) {

**this**.fontTitulo = fontTitulo;

**this**.fontTexto = fontTexto;

**this**.mejoresPuntuaciones = mejoresPuntuaciones;

**this**.volumen = volumen;

**this**.musicaActiva = musicaActiva;

**this**.prota = prota;

}

/\*\* \*/

/\*\* Menu Principal\*/

**void** **dibujarMenu**() {

background(**40**);

// ====== TÍTULO ======

fill(**255**, **215**, **0**); /\*\*Color dorado \*/

textFont(fontTitulo); /\*\*Usar fuente del título \*/

textAlign(CENTER, CENTER); /\*\*Alinear texto al centro \*/

text("Magic Dungeon", width / **2**, **100**); /\*\*Título principal en la parte superior \*/

// ====== MEJORES PUNTUACIONES ====== /\*\* quitar \*/

fill(**200**); // Color gris claro para el título

textFont(fontTexto); // Fuente normal

textAlign(CENTER);

text("MEJORES PUNTUACIONES", width / **2**, **200**);

fill(**255**); // Color blanco para las puntuaciones

**for** (**int** i = **0**; i < min(**5**, mejoresPuntuaciones.size()); i++) {

text((i + **1**) + ". " + mejoresPuntuaciones.get(i), width / **2**, **250** + i \* **40**);

}

/\*\* ====== BOTÓN INICIAR JUEGO ====== \*/

fill(**0**, **255**, **0**); /\*\* verde \*/

rect(width / **2** - **150**, **500**, **300**, **70**, **15**);

fill(**0**); /\*\* texto negro \*/

text("INICIAR JUEGO", width / **2**, **535**);

/\*\* ====== BOTÓN Modo Horda ====== \*/

fill(**0**, **150**, **255**); /\*\* azul \*/

rect(width / **2** - **150**, **600**, **300**, **70**, **15**);

fill(**255**); /\*\* texto blanco \*/

text("MODO HORDA", width / **2**, **635**);

/\*\* ====== BOTÓN SALIR ====== \*/

fill(**255**, **0**, **0**); /\*\* rojo \*/

rect(width / **2** - **150**, **700**, **300**, **70**, **15**);

fill(**0**); /\*\* texto negro \*/

text("SALIR", width / **2**, **735**);

}

/\*\* ====== Modo pausa ====== \*/

**void** **mostrarPausa**() {

fill(**0**, **150**);

rect(**0**, **0**, width, height);

fill(**255**);

textAlign(CENTER, CENTER);

textSize(**48**);

text("PAUSA", width / **2**, height / **2** - **40**);

textSize(**24**);

text("Presiona 'P' para reanudar", width / **2**, height / **2** + **10**);

/\*\* Botón mute/unmute \*/

fill(**255**);

rect(width/**2** - **110**, height/**2** + **80**, **220**, **50**, **8**);

fill(**0**);

textAlign(CENTER, CENTER);

text("Mute/Unmute [M]", width/**2**, height/**2** + **105**);

}

/\*\* ====== Modo Gameover ====== \*/

**void** **mostrarGameOver**() {

background(**0**);

fill(**255**);

textSize(**60**);

textAlign(CENTER, CENTER);

text("GAME OVER", width / **2**, height / **2**);

textSize(**30**);

text("Score: " + prota.getScore(), width / **2**, height / **2** + **60**);

text("Presiona 'R' para reiniciar", width / **2**, height / **2** + **100**);

text("Presiona 'E' para Volver al menu", width / **2**, height / **2** + **150**);

}

/\*\* ====== HUD in-game ====== \*/

**void** **mostrarHUD**() {

pushMatrix();

resetMatrix();

fill(**255**);

textSize(**24**);

textAlign(LEFT);

text("Vidas: " + prota.getVidas(), **20**, **40**);

text("Score: " + prota.getScore(), **20**, **70**);

popMatrix();

}

/\*\* ====== Pantalla nivel 2 ====== \*/

**void** **mostrarNivel2**() {

background(**0**);

fill(**255**);

textSize(**60**);

textAlign(CENTER, CENTER);

text("¡Felicidades! Has alcanzado el Nivel 2", width / **2**, height / **2** - **40**);

textSize(**30**);

text("Presiona n para continuar", width / **2**, height / **2** + **20**);

}

/\*\* ====== Pantalla de fin de juego ====== \*/

**void** **mostrarGameEnd**() {

background(**0**);

fill(**255**);

textSize(**60**);

textAlign(CENTER, CENTER);

text("Haz finalizado la DEMO!!!", width / **2**, height / **2**);

textSize(**30**);

text("Tu Score: " + prota.getScore()+ " Grandioso!!!", width / **2**, height / **2** + **60**);

text("Presiona 'R' para volver al menu", width / **2**, height / **2** + **100**);

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 003 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Creacion de proyectiles | |
| Prioridad: Alta | Riesgo de desarrollo: Alto |
| Estimación:1 dia | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Crear un sistema que permita tanto a enemigos como protagonista dispararse entre si | |
| Criterios de aceptación:  CA1: Se debe crear una clase proyectil  CA2: El protagonista puede disparar  CA3: Los enemigos deben disparar al protagonista apuntandole  CA4: Si el proyectil del protagonista choca con el enemigo, el enemigo desaparece  CA5: Si el proyectil del enemigo choca con el protagonista, el protagonista pierde 1 vida    Bosquejo de situacion del gameplay | |
| Observaciones:  El protagonista como los enemigos crean un arraylist de la clase proyectiles por cada disparo que ejecutan  Tabla EPS (Entrada Proceso Salida) | |
| |  |  | | --- | --- | | Entrada | Entrada del jugador (tecla de disparo), clases Protagonista y Enemigo | | Proceso | Se crean y actualizan proyectiles que pueden colisionar con enemigos o el protagonista; se eliminan proyectiles fuera de pantalla. | | Salida | Sistema de disparos funcional: el protagonista y enemigos pueden atacar y eliminarse mutuamente. | |  |  | | |

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.

**class** **Proyectil** {

**private** PVector posicion; /\*\* posicion del proyectil\*/

**private** PVector velocidad; /\*\* velocidad del proyectil\*/

**private** **float** radio; /\*\* Tamaño del proyectil\*/

**private** color colorProyectil; /\*\* Color del proyectul\*/

**private** **float** dmg; /\*\* Daño\*/

/\*\* Constructores \*/

Proyectil (PVector posicion,color c,PVector velocidad, **float** dmg){

**this**.posicion = posicion.copy();

**this**.velocidad = velocidad.copy();

**this**.radio = **5**;

**this**.colorProyectil = c;

**this**.dmg = dmg;

}

/\*\* \*/

/\*\* actualizacion de velocidad y posicion del proyectil\*/

**void** **update**(**float** deltaTime){

posicion.add(PVector.mult(velocidad, deltaTime));

}

/\*\* dibujado del proyectil\*/

**void** **display**(){

fill(colorProyectil);

noStroke();

circle(posicion.x,posicion.y,radio\***2**);

}

/\*\* borrado del proyectil al salir del lienzo\*/

**boolean** **estaFueraDePantalla**(){

**return** posicion.x < -**2400** || posicion.x > width +**2400** || posicion.y < -**2400** || posicion.y > height +**2400**;

}

/\*\* metodos accesores \*/

**public** PVector **getPosicion**() {

**return** **this**.posicion.copy();

}

**public** **float** **getRadio**() {

**return** **this**.radio;

}

**class** **Protagonista** {

**private** ArrayList<Proyectil> proyectiles; /\*\* Array para la gestion de proyectiles\*/

**private** **int** vidas = **500**; /\*\* Cantidad de vidas del protagonista\*/

**private** **int** score = **0**; /\*\* Puntuacion del protagonista a lo largo del juego\*/

**private** **float** tiempoEntreDisparos = **0.25f**; /\*\* coldown entre disparos\*/

**private** **float** tiempoUltimoDisparo = **0**; /\*\* Temporarizador de disparos \*/

/\*\* Logica de disparo del protagonista + limitacion de balas \*/

**public** **void** **disparar**() {

**if** (tiempoSinDisparo == **0**){

**if** (puedeDisparar) {

PVector direccion = **new** PVector(**0**, -**1**); /\*\* Genera un PVector direccion con una Dirección fija hacia arriba (eje Y negativo) \*/

direccion.mult(**500**); /\*\* Velocidad del proyectil (500 píxeles/segundo) \*/

Proyectil nuevoProyectil = **new** Proyectil( /\*\* Objeto proyectil con los parametros de posicion color, direccion y daño\*/

**new** PVector(posicion.x, posicion.y - radio),

color(**0**, **255**, **0**), /\*\* Color verde para distinguirlas \*/

direccion,

**1** /\*\* Daño del proyectil \*/

);

proyectiles.add(nuevoProyectil);

puedeDisparar = **false**; /\*\* Activacion del coldown\*/

tiempoUltimoDisparo = **0**; /\*\* Reiniciar el temporizador entre disparos \*/

}

**if** (DobleDisparo){

**float** angulo = random(TWO\_PI); // Generar un ángulo aleatorio

PVector direccion = **new** PVector(cos(angulo), sin(angulo)); // Convertir a vector de dirección

direccion.mult(**500**); // Velocidad del proyectil (500 píxeles/segundo)

Proyectil nuevoProyectil = **new** Proyectil(

**new** PVector(posicion.x, posicion.y - radio), // Posición inicial ajustada

color(**255**, **255**, **0**), // Color verde para distinguirlas

direccion,

**1** // Daño del proyectil

);

proyectiles.add(nuevoProyectil);

puedeDisparar = **false**;

tiempoUltimoDisparo = **0**; // Reiniciar el temporizador

}

}

}

/\*\* Funcion para restar vidas del protagonista\*/

**public** **void** **recibirDano**() {

**if** (!invulnerable) { /\*\* Solo reduce vidas si no está invulnerable \*/

vidas--;

}

**public** **int** **getScore**(){

**return** **this**.score;

}

**public** **int** **getVidas**() {

**return** vidas;

}

/\*\* Logica de puntuacion \*/

**public** **void** **aumentarPuntaje**(**int** cantidad) {

score += cantidad; // Aumentar el puntaje

println("score actual del prota: "+ score) ;

}

**public** **void** **reducirPuntaje**(**int** cantidad) {

score -= cantidad; // Reducir el puntaje

println("score actual del prota: "+ score) ;

}

}

**class** **Enemigo** {

**protected** ArrayList<Proyectil> proyectiles; /\*\* \*/

**protected** **int** tiempoDisparo = **0**; /\*\* \*/

**protected** **int** intervaloDisparo = **90**; /\*\* \*/

// El enemigo dispara al protagonista

**protected** **void** **disparar** (Protagonista prota){

tiempoDisparo++;

**if**(tiempoDisparo >= intervaloDisparo){

tiempoDisparo = **0**;

PVector direccion = PVector.sub(prota.getPosicion(), **this**.posicion);

direccion.normalize();

direccion.mult(**100**);

Proyectil nuevoProyectil = **new** Proyectil(**this**.posicion.copy(), color(**0**, **0**, **255**), direccion, **10**);

proyectiles.add(nuevoProyectil);

}

}

// Recarga de proyectiles/eliminar proyectiles fuera del mapa

**protected** **void** **actualizarProyectiles**() {

**for** (**int** i = proyectiles.size() -**1**;i>=**0**;i--){

Proyectil p = proyectiles.get(i);

**if**(p.estaFueraDePantalla()){

proyectiles.remove(i);

}

}

}

**public** ArrayList<Proyectil> **getProyectiles**() {

**return** **this**.proyectiles;

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 004 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Manejador de nivel | |
| Prioridad: Alta | Riesgo de desarrollo: Alto |
| Estimación: 5 dias | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Crear una que maneje la lógica de los niveles | |
| Criterios de aceptación:  El manejador de nivel debe dibujar el escenario  El manejador de nivel debe gestionar el comportamiento del enemigo  El manejador de nivel debe gestionar las colisiones de proyectiles del protagonista y enemigo  El manejador de nivel debe aplicar los efectos de los luckyblocks y eliminarlos  En caso de no completar la condición especial no se podra avanzar de nivel | |
| Observaciones: Cambiar el Estado\_Jugando a Estado\_GameOver cuando las vidas del protagonista sean igual a 0  Los proyectiles que se salgan del lienzo deben eliminarse para cuidar el rendimiento  Por cada enemigo que el protagonista elimina su puntaje aumenta en 100  En el caso de estar en el modo horda el protagonista gana 5 vidas  En el caso de estar en el modo horda por cada enemigo eliminado se generan de 1 a 3 enemigos nuevos de las diferentes variantes existentes  Existe un máximo de 100 enemigos en el campo | |

Tabla EPS (Entrada proceso Salida)

|  |  |
| --- | --- |
| Entrada | Objetos del juego (enemigos, proyectiles, luckyblocks, protagonista, joypad) |
| Proceso | Se actualiza el comportamiento general del nivel: movimiento, colisiones, efectos de cofres, puntuación y condiciones de victoria o derrota. |
| Salida | Gestión completa del flujo del nivel con colisiones, puntuación y control de vidas. |

Interfaz de usuario gráfica, Aplicación, Excel

El contenido generado por IA puede ser incorrecto.

**class** **ManejadorNivel** {

PImage fondo; /\*\* imagen de fondo\*/

ArrayList<Enemigo> enemigos; /\*\* Lista de enemigos en el nivel\*/

ArrayList<Proyectil> proyectilesEnemigos; /\*\* Proyectiles disparados\*/

ArrayList<Lucky> luckyblock; /\*\* bloques especiales\*/

/\*\* Constructor\*/

ManejadorNivel(PImage fondo, ArrayList<Enemigo> enemigos, ArrayList<Proyectil> proyectilesEnemigos, ArrayList<Lucky> luckyblock) {

**this**.fondo = fondo;

**this**.enemigos = enemigos;

**this**.proyectilesEnemigos = proyectilesEnemigos;

**this**.luckyblock = luckyblock;

}

/\*\* Metodo principal de actualizacion\*/

**void** **actualizar**(**float** deltaTime, Protagonista prota, JoyPad joypad, Runnable condicionEspecial) {

image(fondo, **0**, **0**); /\*\* Dibuja la imagen \*/

translate(width / **2** - prota.getPosicion().x, height / **2** - prota.getPosicion().y); /\*\* ajusta la camara para centrar al protagonista\*/

prota.update(deltaTime); /\*\* Actualiza el estado del protagonista\*/

prota.display(protagonista); /\*\* Dibuja al protagonista\*/

/\*\* \*/

/\*\* Actualiza el comportamiento de enemigos\*/

**for** (**int** i = enemigos.size() - **1**; i >= **0**; i--) {

Enemigo e = enemigos.get(i);

e.update(prota, deltaTime); /\*\* Comportamientos, seguir, atacar \*/

e.display(); /\*\* Dibujo del enemigo\*/

**for** (Proyectil p : e.getProyectiles()) { /\*\* Agrega proyectiles a la lista\*/

**if** (!proyectilesEnemigos.contains(p)) {

proyectilesEnemigos.add(p);

}

}

}

/\*\* Actualiza el efecto del luckyblock\*/

**for** (**int** i = luckyblock.size() - **1**; i >= **0**; i--) {

Lucky lucky = luckyblock.get(i);

lucky.display(); /\*\* Dibuja el lucky\*/

**if** (PVector.dist(lucky.posicion, prota.getPosicion()) < **20**) { /\*\* Si el protagonista toca el bloque\*/

lucky.aplicarEfecto(prota); /\*\* Aplica el efecto y elimina el bloque\*/

luckyblock.remove(i);

}

}

/\*\* Actualiza el comportamiento de los proyectiles de enemigos respecto del jugador\*/

**for** (**int** i = proyectilesEnemigos.size() - **1**; i >= **0**; i--) {

Proyectil p = proyectilesEnemigos.get(i);

p.update(deltaTime); /\*\* mover el proyectil \*/

p.display(); /\*\* dibujar el proyectil\*/

/\*\* Colision con el protagonista\*/

**if** (PVector.dist(p.getPosicion(), prota.getPosicion()) < prota.getRadio() + p.getRadio()) {

proyectilesEnemigos.remove(i);

prota.recibirDano(); /\*\* quitar vidas al protagonista\*/

**if** (prota.getVidas() <= **0**) {

estadoActual = ESTADO\_GAMEOVER; /\*\* GameOver al llegar a vidas=0\*/

prota.guardarPuntuacion();

}

}

**if** (p.estaFueraDePantalla()) { /\*\* Si el proyectil esta fuera de pantalla se borra\*/

proyectilesEnemigos.remove(i);

}

}

/\*\* Actualiza el comportamiento de los proyectiles del jugador respecto de enemigos\*/

**for** (Proyectil pJugador : prota.getProyectiles()) {

**for** (**int** i = enemigos.size() - **1**; i >= **0**; i--) {

Enemigo e = enemigos.get(i);

/\*\* Verifica la colision\*/

**if** (PVector.dist(pJugador.getPosicion(), e.getPosicion()) < e.getRadio() + pJugador.getRadio()) {

/\*\* remueve el enemigo\*/

enemigos.remove(i);

/\*\* suma 100 puntos\*/

prota.aumentarPuntaje(**100**);

/\*\* solo en caso de que el modo horda este activo\*/

**if** (HORDA) {

prota.ganarVidas(); /\*\* vidas extra\*/

/\*\* generar una cantidad aleatoria de enemigos nuevos 1,3 \*/

**int** cantidad = **int**(random(**1**, **4**));

/\*\* Limite maximo de enemigos en el campo\*/

**int** limiteMaximo = **100**;

**for** (**int** j = **0**; j < cantidad; j++) {

**if** (enemigos.size() >= limiteMaximo) **break**; /\*\* Generacion de enemigos\*/

/\*\* Nueva posicion aleatoria del enemigo\*/

PVector nuevaPosicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

/\*\* Tipo del enemigo que se generará\*/

**int** tipo = **int**(random(**4**));

**switch** (tipo) {

**case** **0**:

enemigos.add(**new** EnemigoFuerte(nuevaPosicion, enemigo2));

**break**;

**case** **1**:

enemigos.add(**new** EnemigoLento(nuevaPosicion, enemigo1));

**break**;

**case** **2**:

enemigos.add(**new** EnemigoRapido(nuevaPosicion, enemigo3));

**break**;

**default**:

enemigos.add(**new** Enemigo(nuevaPosicion));

**break**;

}

}

}

**break**;

}

}

}

/\*\* Mensaje flotante para la ui\*/

mensajeFlotante.dibujar();

mensajeFlotante.actualizar(deltaTime);

/\*\*Controles dentro del nivel \*/

**if** (joypad.isUpPressed()) prota.mover(**0**, deltaTime);

**if** (joypad.isRightPressed()) prota.mover(**1**, deltaTime);

**if** (joypad.isDownPressed()) prota.mover(**2**, deltaTime);

**if** (joypad.isLeftPressed()) prota.mover(**3**, deltaTime);

/\*\* HUD en pantalla\*/

gestorPantallas.mostrarHUD();

/\*\* CondicionEspecial para poder avanzar al siguiente nivel\*/

**if** (condicionEspecial != **null**) {

condicionEspecial.run();

}

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 005 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Nivel 1 | |
| Prioridad: alta | Riesgo de desarrollo: alta |
| Estimación: 3 dias | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Crear el Primer nivel del videojuego | |
| Criterios de aceptación:  Se debe acceder mediante el menú interactivo  El protagonista cuenta con 500 vidas fijas  Se generan una cantidad de 24 enemigos (8 de cada tipo) se generan y se mueven suavemente por el lienzo del juego  Se generan una cantidad de 15 cofres de la suerte (la suerte se decide aleatoriamente y se genera a lo largo del lienzo del juego  Se alcanza al nivel 2 si y solo si se consigue una puntuación de 500 puntos  El juego acaba si el protagonista no tiene vidas (vidas = 0) o no hay enemigos en pantalla    Captura del Nivel 1 | |
| Observaciones:  Se puede tomar pausa en cualquier momento del videojuego  El manejo del nivel se realiza mediante la clase manejador de nivel, pero el cambio de estado yace en la función -actualizarJuego() | |

Tabla EPS (Entrada Proceso Salida)

|  |  |
| --- | --- |
| Entrada | Datos de ManejadorNivel, puntuación y vidas del protagonista |
| Proceso | Se genera el nivel 1 con enemigos, cofres y puntuación inicial; se evalúa la condición para pasar al nivel 2 o perder. |
| Salida | Nivel inicial jugable con enemigos y cofres; transición al nivel 2 o Game Over. |

/\*\* Logica especifica del nivel 1 \*/

**void** **actualizarJuego**(**float** deltaTime) {

nivel1.actualizar(deltaTime, prota, joypad, () -> { /\*\* se envia deltaTime,protagonista,joypad y () se refiere a la funcion que verifica constantemente que el score del \*/

**if** (prota.getScore() >= **500** && nivel == **1** && HORDA == **false**) { /\*\* protagonista sea menor a 500 y no se encuentre en estado horda de otra manera el estado se cambiaria a nivel2 MENU\*/

nivel = **2**; /\*\* y la generacion de enemigos y niveles se reiniciaria\*/

estadoActual = ESTADO\_NIVEL2MENU;

reiniciarGeneracionNivel2(); /\*\* Se regenera el campo de juego para tener niveles distintos entre partidas \*/

} **else** **if** (enemigos.isEmpty()) { /\*\* De no haber enemigos en campo se entra en gameover\*/

estadoActual = ESTADO\_GAMEOVER;

}

});

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 006 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Nivel 2 | |
| Prioridad: alta | Riesgo de desarrollo: alta |
| Estimación: 3 dias | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Ultimo nivel de la prueba técnica del videojuego | |
| Criterios de aceptación:  El protagonista accede mediante obtener 500 o más puntos en el nivel anterior  El protagonista conserva las vidas del anterior nivel  Se generan una cantidad de 30 enemigos (10 de cada tipo) se generan y se mueven suavemente por el lienzo del juego  Se generan una cantidad de 12 cofres de la suerte (la suerte se decide aleatoriamente y se genera a lo largo del lienzo del juego  Se alcanza el final de la demo si y solo si se consigue una puntuación de 2500 puntos  El juego acaba si el protagonista no tiene vidas (vidas = 0) o no hay enemigos en pantalla    Captura del Nivel 2      Captura de la finalizacion de la Demo | |
| Observaciones: Se limpian los cofres y enemigos del nivel anterior para garantizar variedad de nivel  Se supera el nivel al alcanzar los 2500 puntos y aun contar con vidas | |

Tabla EPS (Tabla Entrada Proceso Salida)

|  |  |
| --- | --- |
| Entrada | Puntuación del jugador, vidas del nivel anterior |
| Proceso | Se generan nuevos enemigos y cofres; se evalúa la condición de victoria (2500 puntos) o derrota. |
| Salida | Segundo nivel del juego y pantalla de finalización de la demo. |

/\*\* Logica del nivel 2\*/

**void** **actualizarNivel2**(**float** deltaTime) {

nivel2.actualizar(deltaTime, prota, joypad, () -> {

**if** (prota.getScore() >= **2500**) {

estadoActual = ESTADO\_CREDITOS;

}

});

}

/\*\* Reset de generacion se enemigos y cofres de la suerte para el nivel 2 \*/

**void** **reiniciarGeneracionNivel2**() {

/\*\* Inicializar las listas por si el jugador llegase a vaciarlas (eliminar todos los enemigos o usar todos los cofres)\*/

**if** (enemigosNivel2 == **null**) enemigosNivel2 = **new** ArrayList<Enemigo>();

**if** (luckyblockNivel2 == **null**) luckyblockNivel2 = **new** ArrayList<Lucky>();

**if** (proyectilesEnemigosNivel2 == **null**) proyectilesEnemigosNivel2 = **new** ArrayList<Proyectil>();

/\*\* Limpiar listas del nivel 2\*/

enemigosNivel2.clear();

luckyblockNivel2.clear();

proyectilesEnemigosNivel2.clear();

/\*\*Generar nuevos lucky blocks para nivel 2 \*/

**for** (**int** j = **0**; j < **20**; j++) {

PVector posicionL = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

**int** tipo = (**int**) random(**2**);

**float** duracion = random(**0**, **15**);

**if** (tipo == **0**) {

luckyblockNivel2.add(**new** LuckyBonus(posicionL, (**int**) random(**0**, **200**), duracion));

} **else** **if** (tipo == **1**) {

luckyblockNivel2.add(**new** LuckyPenalty(posicionL, (**int**) random(**0**, **150**), duracion));

}

luckyblockNivel2.add(**new** Lucky(posicionL, lucky));

}

/\*\*Generar nuevos enemigos para nivel 2 con estadísticas diferentes \*/

**for** (**int** i = **0**; i < **10**; i++) {

PVector posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigosNivel2.add(**new** EnemigoFuerte(posicion, enemigo2));

posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigosNivel2.add(**new** EnemigoLento(posicion, enemigo1));

posicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

enemigosNivel2.add(**new** EnemigoRapido(posicion, enemigo3));

}

}

/\*\* Reset de del videojuego en caso de llegar al estado\_creditos o perder en el nivel 2, etc \*/

**void** **reiniciarJuego**() {

nivel = **1**; /\*\* volver al nivel 1 \*/

HORDA = **false**; /\*\* Desactivar el modo horda \*/

proyectilesEnemigos.clear(); /\*\*Limpiar proyectiles enemigos \*/

enemigos.clear(); /\*\*Limpiar enemigos \*/

luckyblock.clear(); /\*\*Limpiar lucky blocks \*/

setup();

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 007 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Modo Horda | |
| Prioridad: alta | Riesgo de desarrollo: alta |
| Estimación: 3 dias | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Modo principal de juego donde el jugador se verá envuelto en un “infierno” de balas donde tendrá que conseguir su máxima puntuación | |
| Criterios de aceptación:  El modo de juego es de duración ilimitada, por lo tanto por cada eliminación de enemigo únicamente en este modo ganará 5 vidas, con un máximo de 1200 posibles  Por cada eliminación de enemigos pueden generarse de 1 a 3 enemigos de diferente tipo aleatoriamente en el campo por lo que siempre existirán enemigos y balas hacia el protagonista    Captura del modo Horda | |
| Observaciones: Se puede tomar pausa en cualquier momento del videojuego  El manejo del nivel se realiza mediante la clase manejador de nivel, pero el cambio de estado yace en la función -actualizarHorda()  El modo horda se activa y desactiva mediante un valor booleano | |

Tabla EPS (Entrada Proceso Salida)

|  |  |
| --- | --- |
| Entrada | Activación del modo horda, datos del protagonista y posición del enemigo enemigos |
| Proceso | Se activa el modo con generación constante de enemigos y aumento de vidas por cada eliminación. |
| Salida | Modo de juego infinito con dificultad progresiva o GameOver |

/\*\* Logica del nivel horda\*/

**void** **actualizarHorda**(**float** deltaTime){

HORDA = **true**;

nivel1.actualizar(deltaTime, prota, joypad, () -> {});

}

**class** **ManejadorNivel** {

/\*\* solo en caso de que el modo horda este activo\*/

**if** (HORDA) {

prota.ganarVidas(); /\*\* vidas extra\*/

/\*\* generar una cantidad aleatoria de enemigos nuevos 1,3 \*/

**int** cantidad = **int**(random(**1**, **4**));

/\*\* Limite maximo de enemigos en el campo\*/

**int** limiteMaximo = **100**;

**for** (**int** j = **0**; j < cantidad; j++) {

**if** (enemigos.size() >= limiteMaximo) **break**; /\*\* Generacion de enemigos\*/

/\*\* Nueva posicion aleatoria del enemigo\*/

PVector nuevaPosicion = **new** PVector(random(**0**, **2400**), random(**0**, **1600**));

/\*\* Tipo del enemigo que se generará\*/

**int** tipo = **int**(random(**4**));

**switch** (tipo) {

**case** **0**:

enemigos.add(**new** EnemigoFuerte(nuevaPosicion, enemigo2));

**break**;

**case** **1**:

enemigos.add(**new** EnemigoLento(nuevaPosicion, enemigo1));

**break**;

**case** **2**:

enemigos.add(**new** EnemigoRapido(nuevaPosicion, enemigo3));

**break**;

**default**:

enemigos.add(**new** Enemigo(nuevaPosicion));

**break**;

}

}

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 008 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Diseño de personajes y entorno | |
| Prioridad: media-baja | Riesgo de desarrollo: bajo |
| Estimación: 1 dia | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Se establecerán la imagen de fondo de los niveles, menú de inicio,pausa,final y de los diferentes enemigos | |
| Criterios de aceptación:  Cada nivel deberá tener una imagen propia  El protagonista como los diferentes enemigos deben contar con una imagen que los caracterice  El menú principal debe contar con un logo e imágenes para acceder a diferentes modos    Captura de la pantalla de felicitaciones hacia el nivel 2    Fondo del escenario 1   |  |  | | --- | --- | |  | Cofre de la suerte | |  | Protagonista | |  | Enemigo lento | |  | Enemigo fuerte | |  | Enemigo rapido |     Fondo del escenario 2 | |
| Observaciones:  Según el tipo de enemigo el tamaño del mismo puede variar  Se precargan las imágenes y fuentes en el setup  El diseño de las pantallas yace en la clase gestorPantallas | |

Tabla EPS (Entrada Proceso Salida)

|  |  |
| --- | --- |
| Entrada | Archivos de imágenes (fondo1.png, enemigo1.png, etc.) y fuentes |
| Proceso | Se cargan y redimensionan imágenes; se asocian a enemigos, niveles y protagonista. |
| Salida | Aspecto visual del juego definido (fondos, personajes, cofres, menús). |

**private** PImage fondo1; /\*\* Imagen del nivel 1\*/

**private** PImage fondo2; /\*\* Imagen del nivel 2\*/

**private** PImage lucky; /\*\* Imagen del cofre de la suerte\*/

**private** PImage enemigo1; /\*\* Imagen del enemigo 1\*/

**private** PImage enemigo2; /\*\* Imagen del enemigo 2\*/

**private** PImage enemigo3; /\*\* Imagen del enemigo 3\*/

**private** PImage protagonista; /\*\* Imagen del protagonista\*/

fondo1 = loadImage("fondo1.png"); /\*\* Imagen del nivel 1\*/

fondo2 = loadImage("fondo2.png"); /\*\* Imagen del nivel 2\*/

lucky = loadImage("lucky.png"); /\*\* Imagen correspondiente al cofre de la suerte\*/

enemigo1 = loadImage("enemigo1.png"); /\*\* Imagen correspondiente al enemigo lento\*/

enemigo2= loadImage("enemigo2.png"); /\*\* Imagen correspondiente al enemigo fuerte\*/

enemigo3= loadImage("enemigo3.png"); /\*\* Imagen correspondiente al enemigo rapido\*/

protagonista = loadImage("protagonista.png"); /\*\* Imagen correspondiente al protagonista\*/

protagonista.resize(**50**,**70**); /\*\* Redimenzionamiento del protagonista\*/

enemigo2.resize(**80**,**80**); /\*\* Redimenzionamiento del enemigo\*/

enemigo3.resize(**50**, **60**); /\*\* Redimenzionamiento del enemigo\*/

lucky.resize(**50**, **50**); /\*\* Redimenzionamiento del cofre\*/

enemigo1.resize(**50**, **60**); /\*\* Redimenzionamiento del enemigo\*/

fondo1.resize(width, height); /\*\* Redimenzionamiento del nivel1\*/

fondo2.resize(width, height); /\*\* Redimenzionamiento del nivel2\*/

**private PFont fontTitulo; /\*\* Fuente del titulo\*/**

**private PFont fontTexto; /\*\* Fuente de textos\*/**

**private** PImage fondo1; /\*\* Imagen del nivel 1\*/

**private** PImage fondo2; /\*\* Imagen del nivel 2\*/

**private** PImage lucky; /\*\* Imagen del cofre de la suerte\*/

**private** PImage enemigo1; /\*\* Imagen del enemigo 1\*/

**private** PImage enemigo2; /\*\* Imagen del enemigo 2\*/

**private** PImage enemigo3; /\*\* Imagen del enemigo 3\*/

**private** PImage protagonista; /\*\* Imagen del protagonista\*/

**void** **setup**() {

fondo1 = loadImage("fondo1.png"); /\*\* Imagen del nivel 1\*/

fondo2 = loadImage("fondo2.png"); /\*\* Imagen del nivel 2\*/

lucky = loadImage("lucky.png"); /\*\* Imagen correspondiente al cofre de la suerte\*/

enemigo1 = loadImage("enemigo1.png"); /\*\* Imagen correspondiente al enemigo lento\*/

enemigo2= loadImage("enemigo2.png"); /\*\* Imagen correspondiente al enemigo fuerte\*/

enemigo3= loadImage("enemigo3.png"); /\*\* Imagen correspondiente al enemigo rapido\*/

protagonista = loadImage("protagonista.png"); /\*\* Imagen correspondiente al protagonista\*/

protagonista.resize(**50**,**70**); /\*\* Redimenzionamiento del protagonista\*/

enemigo2.resize(**80**,**80**); /\*\* Redimenzionamiento del enemigo\*/

enemigo3.resize(**50**, **60**); /\*\* Redimenzionamiento del enemigo\*/

lucky.resize(**50**, **50**); /\*\* Redimenzionamiento del cofre\*/

enemigo1.resize(**50**, **60**); /\*\* Redimenzionamiento del enemigo\*/

fondo1.resize(width, height); /\*\* Redimenzionamiento del nivel1\*/

fondo2.resize(width, height); /\*\* Redimenzionamiento del nivel2\*/

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 008 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Enemigo rápido, Enemigo Fuerte, Enemigo lento | |
| Prioridad: media-baja | Riesgo de desarrollo: bajo |
| Estimación: 1 dia | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Se crean clases heredadas de enemigo para aumentar la variedad del videojuego | |
| Criterios de aceptación:  Cada enemigo cuenta con una imagen distinta característica  Cada enemigo cuenta con un intervalo de disparo distinto  Cada enemigo cuenta con una velocidad diferente  (poner imagen) | |
| Observaciones: El enemigo fuerte cuenta con un tamaño considerablemente más grande  El enemigo lento tiene un intervalo de disparo de 190 frames  El enemigo rápido cuenta con un intervalo de disparo de 60 frames | |

Tabla EPS (entrada Proceso Salida)

|  |  |
| --- | --- |
| Entrada | Parámetros de clase base Enemigo |
| Proceso | Se crean clases hijas con distintos comportamientos (velocidad, tamaño, intervalo de disparo). |
| Salida | Variedad de enemigos con características propias. |

Diagrama

El contenido generado por IA puede ser incorrecto.

/\*\* Clase heredada de enemigo \*/

**class** **EnemigoFuerte** **extends** Enemigo {

**private** PImage enemigo2;

**public** **EnemigoFuerte**(PVector posicion,PImage enemigo2) {

**super**(posicion); /\*\* Se hereda su posicion \*/

**this**.radio = **30**; /\*\* Tamaño aumentado\*/

**this**.colorEnemigo = color(**255**, **165**, **0**); /\*\* color de versiones anteriores\*/

**this**.intervaloDisparo = **120**; /\*\* Intervalo de disparo aumentado\*/

**this**.enemigo2 = enemigo2;

}

**@Override**

**protected** **void** **disparar**(Protagonista prota) {

/\*\* Logica particular del disparo del enemigo fuerte\*/

tiempoDisparo++;

**if** (tiempoDisparo >= intervaloDisparo) {

tiempoDisparo = **0**;

PVector direccion = PVector.sub(prota.getPosicion(), **this**.posicion);

direccion.normalize(); /\*\* Se convierte en un vector unitario\*/

direccion.mult(**200**); /\*\* Velocidad del disparo aumentada\*/

Proyectil nuevoProyectil = **new** Proyectil(**this**.posicion.copy(), color(**255**, **0**, **0**), direccion, **20**); /\*\* \*/

proyectiles.add(nuevoProyectil); /\*\* \*/

}

}

**@Override**

**public** **void** **display**() {

/\*\* Metodo alterado para dibujar al enemigo fuerte\*/

image(enemigo2, posicion.x - enemigo2.width / **2**, posicion.y - enemigo2.height / **2**);

}

}

/\*\* Variante de enemigo, enemigo lento\*/

**class** **EnemigoLento** **extends** Enemigo {

**private** PImage enemigo1;

**public** **EnemigoLento**(PVector posicion,PImage enemigo1) {

**super**(posicion); /\*\* Se hereda la posicion generada\*/

**this**.velocidad = **new** PVector(random(**1**, **2**), random(**1**, **2**)); /\*\* velocidad alterada a una más lenta \*/

**this**.colorEnemigo = color(**0**, **0**, **255**); /\*\* Color base azul \*/

**this**.enemigo1 = enemigo1;

**this**.intervaloDisparo = **190**; /\*\* Intervalo de disparo alterado, uno mucho más lento \*/

}

**@Override**

**public** **void** **display**() {

/\*\* Metodo de dibujo alterado con la imagen particular del enemigo lento\*/

image(enemigo1, posicion.x - enemigo1.width / **2**, posicion.y - enemigo1.height / **2**);

}

}

/\*\* Variante de enemigo enemigo Rapido \*/

**class** **EnemigoRapido** **extends** Enemigo {

**private** PImage enemigo3;

**public** **EnemigoRapido**(PVector posicion, PImage enemigo3) {

**super**(posicion); /\*\* Hereda la posicion de la clase padre \*/

**this**.velocidad = **new** PVector(random(**3**, **5**), random(**3**, **5**)); /\*\* Velocidad aumentada \*/

**this**.colorEnemigo = color(**0**, **255**, **0**); /\*\* Color base distintivo\*/

**this**.intervaloDisparo = **60**; /\*\*Intervalo de disparo más bajo\*/

**this**.enemigo3 = enemigo3;

}

**@Override**

**public** **void** **display**() {

/\*\* Metodo de dibujo alterado con la imagen particular del enemigo rapido\*/

image(enemigo3, posicion.x - enemigo3.width / **2**, posicion.y - enemigo3.height / **2**); // Centrar la imagen

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 009 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: LuckyBonus y Lucky Penalty | |
| Prioridad: media | Riesgo de desarrollo: bajo |
| Estimación: 1 dia | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Se crean los efectos de los luckyblocks tanto efectos positivos como efectos negativos | |
| Criterios de aceptación:  Se deben crear 3 efectos positivos para el jugador  Se deben crear 3 efectos negativos para el jugador  La aplicación del efecto debe ser aleatoria según se tome el cofre de la suerte    Doble Disparo in-game (Disparo verde el principal, disparo amarillo el secundario) | |
| Observaciones:  Se creo una bonificación de doble disparo e invencibilidad para el protagonista la cual se maneja en la clase protagonista  La tercer bonificación aumenta la puntuación en 1000 puntos directamente  Siempre se informará al jugador de la bonificación o penalización obtenida por medio de un mensaje en pantalla de la clase MensajeFlotante  Las penalizaciones constan de:  Reducción drástica de puntuación del protagonista (2000 puntos)  Si la puntuación del protagonista es 0 y se adquiere una penalización es valido obtener una puntuación negativa  Cese de disparo del protagonista y de estar activa la función de doble disparo se apaga también. (Se podrá volver a activar de encontrar el cofre adecuado)  Aumento de enemigos en pantalla en una cantidad de 1 a 12 esparcidos por el mapa mediante un método que yace en la raíz. | |

**Tabla EPS (Entrada Proceso Salida)**

|  |  |
| --- | --- |
| **Entrada** | **Colisión del protagonista con un cofre de la suerte** |
| **Proceso** | **Se aplica un efecto aleatorio (bonus o penalización) modificando vidas, disparos, puntaje o cantidad de enemigos.** |
| **Salida** | **Sistema de cofres con efectos positivos o negativos y mensaje en pantalla.** |

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**class** **Lucky** {

**private** PVector posicion; /\*\* Posicion del cofre \*/

**private** PImage lucky;/\*\* Imagen del cofre\*/

**public** **Lucky**() {

**this**.posicion = **new** PVector(**0**, **0**);

}

**public** **Lucky**(PVector posicion, PImage lucky) { /\*\* Constructor \*/

**this**.posicion = posicion;

**this**.lucky = lucky;

}

**public** **void** **display**() { /\*\* Dibujo del cofre en pantalla \*/

image(lucky,posicion.x,posicion.y);

}

**public** **void** **aplicarEfecto** (Protagonista Prota){ /\*\* Metodo general para aplicar el efecto\*/

}

}

**class** **LuckyBonus** **extends** Lucky {

**private** **int** bonificacion; /\*\* numero de puntaje aleatorio aumentado al protagonista \*/

**private** **float** duracion; /\*\* Duracion de efectos\*/

**public** **LuckyBonus**(PVector posicion, **int** bonificacion, **float** duracion) { /\*\* Constructor\*/

**super**(posicion, lucky);

**this**.bonificacion = bonificacion;

**this**.duracion = duracion;

}

**@Override**

**public** **void** **aplicarEfecto**(Protagonista prota) {

prota.aumentarPuntaje(bonificacion); /\*\* Aumento en puntaje\*/

**int** bonus = (**int**) random(**3**); /\*\* Numero aleatorio para determinar la bonificacion a usar\*/

**if** (bonus == **0**) {

prota.setDobleDisparo(); /\*\* Bonificacion de doble disparo del protagonista\*/

mostrarMensaje("¡Doble Disparo!", **3.0f**, color(**0**, **255**, **0**)); /\*\* Mensaje de bonus \*/

} **else** **if** (bonus == **1**) {

prota.setInvulnerable(duracion); /\*\*Bonificacion de invulnerabilidad y duracion de la misma\*/

mostrarMensaje("¡Eres invulnerable por " + duracion + " segundos!", **3.0f**, color(**0**, **255**, **0**)); /\*\* Mensaje de bonus \*/

}

**else** **if** (bonus == **2**){

prota.aumentarPuntaje(**1000**); /\*\* Aumento drastico de puntaje \*/

mostrarMensaje("¡Eres Increible tu puntuacion es de " + prota.getScore() + " ", **3.0f**, color(**0**, **255**, **0**)); /\*\* Mensaje de bonus con la nueva puntuacion \*/

}

}

**@Override**

**public** **void** **display**() { /\*\* Metodo de dibujo\*/

fill(**0**, **255**, **0**);

**super**.display();

}

}

**class** **LuckyPenalty** **extends** Lucky {

**private** **int** penalizacion; /\*\* Numero en el que el puntaje del protagonista puede reducirse\*/

**private** **float** duracion; /\*\* duracion de la penlizacion\*/

**public** **LuckyPenalty**(PVector posicion, **int** penalizacion, **float** duracion){ /\*\* Constructor\*/

**super** (posicion,lucky);

**this**.penalizacion = penalizacion;

**this**.duracion = duracion;

}

**@Override**

**public** **void** **aplicarEfecto**(Protagonista prota) {

prota.reducirPuntaje(penalizacion); /\*\* Reduccion de puntaje \*/

**int** debuff = (**int**) random (**3**); /\*\* \*/

**if** (debuff == **0**){

prota.setDisparoFalse(duracion); /\*\* Apaga la funcion de disparo del protagonista \*/

prota.setDobleDisparoOFF(); /\*\* Apagar doble disparo\*/

mostrarMensaje("¡Sin disparos!", **3.0f**, color(**0**, **255**, **0**)); /\*\* Mensaje de penalizacion \*/

}

**if** (debuff == **1**){

prota.reducirPuntaje(**2000**); /\*\* Reduccion de puntaje drastica\*/

mostrarMensaje("Hemos reducido tu puntaje para alargar el juego artificialmente ", **3.0f**, color(**0**, **255**, **0**)); /\*\* Mensaje de penalizacion \*/

}

**if** (debuff == **2**){

**int** cantidadEnemigos = (**int**) random(**1**, **12**); /\*\* Aumenta la cantidad de enemigos aleatoriamente entre 1 y 12 \*/

agregarEnemigos(cantidadEnemigos); /\*\* Llamado a la funcion de raiz, agregar enemigos \*/

mostrarMensaje("aumente la cantidad enemigos, estaba muy facil!!", **3.0f**, color(**0**, **255**, **0**)); /\*\* Mensaje de penalizacion\*/

}

}

**@Override**

**public** **void** **display**() {

fill(**255**, **0**, **0**); /\*\* Dibujo del cofre\*/

**super**.display();

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 010 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Gestor de Audio | |
| Prioridad: media-baja | Riesgo de desarrollo: bajo |
| Estimación: 1 dia | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Se deberán implementar audios para ambientar el videojuego como música de fondo, música para el menú, música para el gameover, música para la pantalla hacia el nivel 2 | |
| Criterios de aceptación:  CA1: Debe haber una canción para el menú de pausa  CA2: Debe haber una canción para el nivel 1  CA3: Debe haber una canción para el nivel 2  CA3: Debe haber una canción para la pantalla de felicitaciones al nivel 2 | |
| Observaciones:  Las canciones se administran mediante la clase manejador de audio  Las canciones se encuentran en loop  Las canciones pueden mutearse con la tecla m  Las canciones se pausan al entrar en pausa | |

Tabla EPS (Entrada Proceso Salida)

|  |  |
| --- | --- |
| Entrada | Archivos de audio (menu.wav, nivel1.mp3, etc.), estado del juego |
| Proceso | Se controla reproducción, volumen y muteo de pistas según el estado del juego. |
| Salida | Música contextual en cada parte del juego y control de sonido. |

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.

**class** **GestorAudio** {

Minim minim; /\*\* \*/

AudioPlayer musicaMenu; /\*\* atributo correspondiente a la musica del menu \*/

AudioPlayer musicaJugando; /\*\* atributo correspondiente a la musica del nivel 1\*/

AudioPlayer musicaNivel2; /\*\* atributo correspondiente a la musica del nivel 2\*/

AudioPlayer musicavictoria; /\*\* atributo correspondiente a la musica de la victoria\*/

AudioPlayer musicaderrota; /\*\* \*/

**float** volumen = **0.7**; /\*\* atributo del volumen\*/

**boolean** muteado = **false**; /\*\* atributo del mute\*/

**int** estadoActual = -**1**; /\*\* atributo Para controlar cambios\*/

/\*\* Carga de archivos de audio\*/

GestorAudio(PApplet app) {

minim = **new** Minim(app);

musicaMenu = minim.loadFile("menu.wav");

musicaJugando = minim.loadFile("nivel1.mp3");

musicaNivel2 = minim.loadFile("nivel2.wav");

musicavictoria = minim.loadFile("victoria.mp3");

musicaderrota = minim.loadFile("derrota.mp3");

setVolumen(volumen);

}

/\*\* Método auxiliar para detener y reiniciar una pista \*/

**void** **detener**(AudioPlayer m) {

**if** (m != **null**) {

m.pause();

m.rewind();

}

}

/\*\* Metodo actualizar con switch del estado de la musica\*/

**void** **actualizar**(**int** nuevoEstado) {

**if** (estadoActual == nuevoEstado) **return**; /\*\* almacenar el estado actual \*/

/\*\* Switch para detener musica\*/

**switch** (estadoActual) {

**case** ESTADO\_MENU: detener(musicaMenu); **break**;

**case** ESTADO\_JUGANDO: detener(musicaJugando); **break**;

**case** ESTADO\_NIVEL2: detener(musicaNivel2); **break**;

**case** ESTADO\_NIVEL2MENU: detener(musicavictoria); **break**;

**case** ESTADO\_GAMEOVER: detener(musicaderrota); **break**;

**case** ESTADO\_HORDA: detener(musicaJugando); **break**;

**case** ESTADO\_CREDITOS: detener(musicavictoria); **break**;

}

/\*\* Iniciar nueva musica \*/

**if** (!muteado) {

**switch**(nuevoEstado) {

**case** ESTADO\_MENU:

musicaMenu.rewind(); /\*\* musica rebobinada\*/

musicaMenu.loop(); /\*\* musica en loop\*/

**break**;

**case** ESTADO\_JUGANDO:

musicaJugando.rewind();

musicaJugando.loop();

**break**;

**case** ESTADO\_NIVEL2:

musicaNivel2.rewind();

musicaNivel2.loop();

**break**;

**case** ESTADO\_HORDA:

musicaJugando.rewind();

musicaJugando.loop();

**break**;

**case** ESTADO\_CREDITOS:

musicavictoria.rewind();

musicavictoria.loop();

**break**;

**case** ESTADO\_NIVEL2MENU:

musicavictoria.rewind();

musicavictoria.loop();

**break**;

}

}

estadoActual = nuevoEstado; /\*\* almacenar estado\*/

}

**void** **setVolumen**(**float** vol) { /\*\* metodo para establecer el volumen \*/

volumen = constrain(vol, **0**, **1**);

**float** dB = map(volumen, **0**, **1**, -**40**, **0**);

musicaMenu.setGain(dB);

musicaJugando.setGain(dB);

musicaNivel2.setGain(dB);

musicavictoria.setGain(dB);

musicaderrota.setGain(dB);

}

**void** **toggleMute**(**boolean** muteado) { /\*\* metodo para activar el mute o desactivarlo \*/

**this**.muteado = muteado;

**if** (muteado) {

musicaMenu.mute();

musicaJugando.mute();

musicaNivel2.mute();

musicaderrota.mute();

musicavictoria.mute();

} **else** {

musicaMenu.unmute();

musicaJugando.unmute();

musicaNivel2.unmute();

musicaderrota.unmute();

musicavictoria.unmute();

actualizar(estadoActual); // reanudar la música del estado actual

}

}

**void** **detener**() { /\*\* Metodo para detener la musica\*/

musicaMenu.close();

musicaJugando.close();

musicaNivel2.close();

musicaderrota.close();

musicavictoria.close();

minim.stop();

}

**void** **detenerMusicaActual**() { /\*\* switch para detener la musica que se esta escuchando \*/

**switch** (estadoActual) {

**case** ESTADO\_MENU: detener(musicaMenu); **break**;

**case** ESTADO\_JUGANDO: detener(musicaJugando); **break**;

**case** ESTADO\_NIVEL2: detener(musicaNivel2); **break**;

**case** ESTADO\_HORDA: detener(musicaJugando); **break**;

**case** ESTADO\_CREDITOS: detener(musicavictoria); **break**;

**case** ESTADO\_GAMEOVER: detener(musicaderrota); **break**;

**case** ESTADO\_NIVEL2MENU: detener(musicavictoria); **break**;

}

}

}

Historia de usuario

|  |  |
| --- | --- |
| Código: 011 | Usuario: Lautaro |
| Nombre de la Historia de Usuario: Mensajes Flotantes con información para el jugador | |
| Prioridad: media-baja | Riesgo de desarrollo: bajo |
| Estimación: 1 dia | iteración Asignada: 1 |
| Responsable: Espinosa Lautaro Eduardo | |
| Descripción: Se envían mensajes flotantes con el efecto aplicado de los cofres de la suerte | |
| Criterios de aceptación:  El mensaje debe informar el efecto aplicado por el cofre de la suerte | |
| Observaciones:  En la tercer opción de Luckybonus el mensaje flotante informa de la nueva puntuación del personaje | |

Tabla EPS (Entrada Proceso Salida)

|  |  |
| --- | --- |
| Entrada | Efectos de cofres o eventos en el juego |
| Proceso | Se muestran mensajes flotantes temporales indicando efectos o información relevante. |
| Salida | Retroalimentación visual informativa para el jugador. |

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

**class** **MensajeFlotante** {

**private** String texto; /\*\* Atributo del cadena de texto\*/

**private** **float** tiempoRestante; /\*\* Atributo del tiempo del msj\*/

**private** PFont fuente; /\*\* atributo de la fuente\*/

**private** **int** colorTexto; /\*\* atributo del color del texto\*/

**public** **MensajeFlotante**(PFont fuente) { /\*\* constructor\*/

**this**.fuente = fuente;

**this**.texto = "";

**this**.tiempoRestante = **0**;

**this**.colorTexto = color(**250**, **250**, **250**);

}

**public** **void** **mostrar**(String mensaje, **float** duracion, **int** colores) { /\*\* constructor\*/

**this**.texto = mensaje;

**this**.tiempoRestante = duracion;

**this**.colorTexto = colores;

}

**public** **void** **actualizar**(**float** deltaTime) { /\*\* metodo para establecer el tiempo del mensaje\*/

**if** (tiempoRestante > **0**) {

tiempoRestante -= deltaTime;

}

}

**public** **void** **dibujar**() { /\*\* metodo para dibujar el mensaje\*/

**if** (tiempoRestante > **0**) {

pushMatrix();

resetMatrix();

fill(colorTexto);

textFont(fuente);

textAlign(CENTER, CENTER);

text(texto, width / **2**, **50**);

popMatrix();

}

}

}