

Informe de ampliaciones

:)



Ikram El Mabrouk Morhnane

- Ampliación 1: Nuevo tipo de enemigo

Cree una clase abstracta `EnemyPadre` que sirve como base para los diferentes tipos de enemigos. Esta clase contiene atributos y métodos comunes, como la animación ,aunque en mi caso no hubo animación , y el comportamiento básico de actualización y dibujo.

```

NavesL4.cpp (Ámbito global)
1 #include "EnemyB.h"
2
3
4 // Constructor
5 EnemyB::EnemyB(float x, float y, Game* game)
6   : EnemyPadre("res/marciano.png", x, y, 40, 40, game) {
7
8   animation = new Animation("res/marciano.png", width, height,
9     120, 42, 4, 3, game);
10  vx = -2;
11 }
12
13 void EnemyB::update() {
14   x += -2.0f; // avanza a la izquierda
15   angle += 0.2f; // se usa como temporizador del parpadeo
16   shootCounter++;
17 }
18
19 void EnemyB::draw() {
20   // se dibuja solo cuando sin(angle) > 0 +esto lo hice para el efecto parpadeo :
21   if (sin(angle) > 0) {
22     animation->draw(x, y);
23   }
24 }
25 
```



```

NavesL4.cpp (Ámbito global)
1 #include "EnemyA.h"
2
3
4 // Constructor
5 EnemyA::EnemyA(float x, float y, Game* game)
6   : EnemyPadre("res/enemigo.png", x, y, 36, 40, game) {
7
8   animation = new Animation("res/enemigo_movimiento.png", width, height,
9     108, 40, 6, 3, game);
10  vx = -1;
11 }
12
13 void EnemyA::update() {
14   x += vx;
15   shootCounter++;
16 }
17
18 void EnemyA::draw() {
19   animation->draw(x, y);
20 } 
```



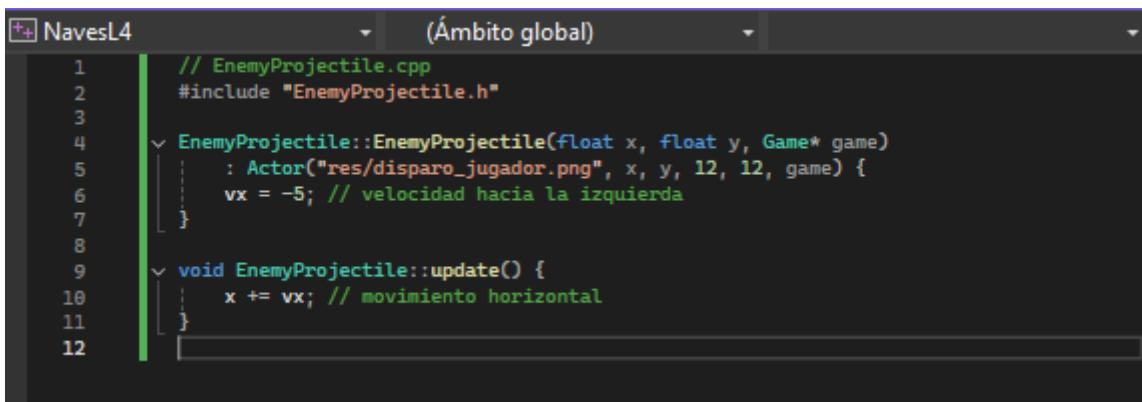
```

NavesL4.h (Ámbito global)
1 #pragma once
2
3 #include "Actor.h"
4 #include "Animation.h"
5 #include "EnemyProjectile.h"
6
7 class EnemyPadre : public Actor {
8 public:
9   EnemyPadre(string filename, float x, float y, int width, int height, Game* game)
10
11   virtual void update() = 0;
12   virtual void draw() override = 0;
13
14   Animation* animation = nullptr;
15   int shootCounter = 0; // contador para saber cuándo disparar
16
17   // Método para crear un disparo
18   virtual EnemyProjectile* shoot();
19 }; 
```

- **Ampliación 2: Enemigos con capacidad de disparo**

Se añadió a los enemigos la capacidad de disparar proyectiles hacia el jugador. Cada enemigo genera un disparo cada N iteraciones del juego, controladas mediante un contador interno (shootCounter).

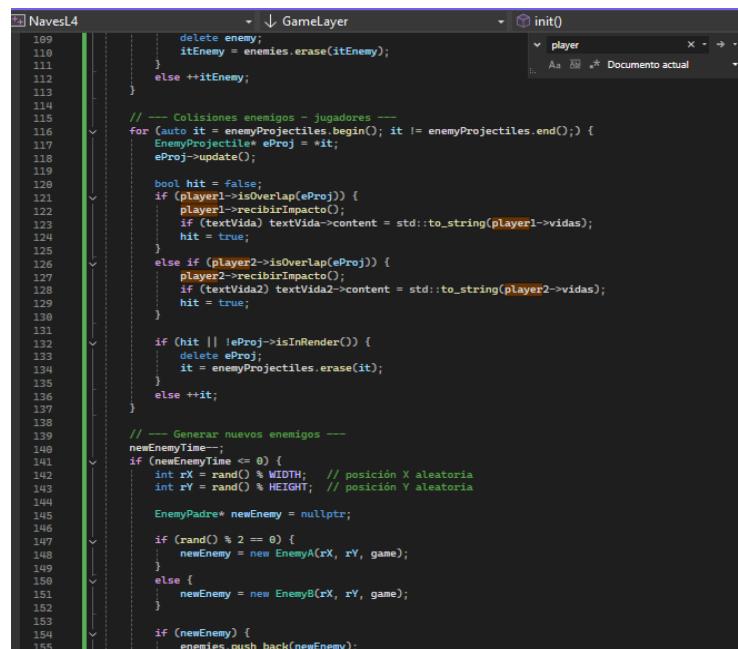
- Cada enemigo hereda de EnemyPadre y utiliza el método shoot() para generar proyectiles (EnemyProjectile).
- Los proyectiles se actualizan en cada iteración del juego y pueden colisionar con el jugador.
- La frecuencia de disparo se controla mediante un contador que se incrementa en cada update(). Cuando alcanza un valor definido (por ejemplo, cada 100 frames), el enemigo dispara.
- Los proyectiles se agregan a la lista de enemyProjectiles y se dibujan en pantalla.



```

1 // EnemyProjectile.cpp
2 #include "EnemyProjectile.h"
3
4 EnemyProjectile::EnemyProjectile(float x, float y, Game* game)
5     : Actor("res/disparo_jugador.png", x, y, 12, 12, game) {
6     vx = -5; // velocidad hacia la izquierda
7 }
8
9 void EnemyProjectile::update() {
10     x += vx; // movimiento horizontal
11 }
12

```



```

109         delete enemy;
110         itEnemy = enemies.erase(itEnemy);
111     } else ++itEnemy;
112 }
113
114 // --- Colisiones enemigos - jugadores ---
115 for (auto it = enemyProjectiles.begin(); it != enemyProjectiles.end(); ) {
116     EnemyProjectile* eProj = *it;
117     eProj->update();
118
119     bool hit = false;
120     if (player1->isOverlap(eProj)) {
121         player1->recibirImpacto();
122         if (textVida) textVida->content = std::to_string(player1->vidas);
123         hit = true;
124     }
125     else if (player2->isOverlap(eProj)) {
126         player2->recibirImpacto();
127         if (textVida2) textVida2->content = std::to_string(player2->vidas);
128         hit = true;
129     }
130
131     if (hit || !eProj->isInRender()) {
132         delete eProj;
133         it = enemyProjectiles.erase(it);
134     } else ++it;
135 }
136
137 // --- Generar nuevos enemigos ---
138 newEnemyTime--;
139 if (newEnemyTime <= 0) {
140     int rX = rand() % WIDTH; // posición X aleatoria
141     int rY = rand() % HEIGHT; // posición Y aleatoria
142
143     EnemyPadre* newEnemy = nullptr;
144
145     if (rand() % 2 == 0) {
146         newEnemy = new EnemyA(rX, rY, game);
147     }
148     else {
149         newEnemy = new EnemyB(rX, rY, game);
150     }
151
152     if (newEnemy) {
153         enemies.push_back(newEnemy);
154     }
155 }

```

- **Ampliación 3: Jugador con Vida**

- Se implementó un sistema de vidas para el jugador:
- El jugador empieza con 3 vidas (vidas).
- Cada vez que un enemigo o un proyectil enemigo lo alcanza, pierde 1 vida.
- Las vidas restantes se muestran en pantalla mediante un texto (textVida).
- El juego no se reinicia automáticamente al perder vida; solo se reduce la vida y se elimina el enemigo o proyectil que causó el impacto.

```

NavesL4                                         (Ámbito global)

1   #pragma once
2
3   #include "Actor.h"
4   #include "Projectile.h"
5   #include "Audio.h"
6   #include <string>
7
8   class Player : public Actor
9   {
10  public:
11      Player(float x, float y, Game* game);
12
13      Projectile* shoot();
14      void update() override;
15      void moveX(float axis);
16      void moveY(float axis);
17
18      Audio* audioShoot;
19      int shootCadence = 30;
20      int shootTime = 0;
21
22      // --- NUEVO: vidas del jugador ---
23      int vidas = 3; // empieza con 3 vidas
24      void recibirImpacto(); // decrementa vida
25
26      // --- NUEVO: cambio de nave ---
27      void setShipType(int type);
28
29  private:
30      float moveSpeed = 3.0f; // velocidad de movimiento, cambia según la nave
31      int shipType = 1; // tipo de nave actual
32  };
33

```

```

NavesL4                                         GameLayer
10 #include "Text.h"
11 #include "Audio.h"
12 #include <list>
13
14 class GameLayer : public Layer
15 {
16 public:
17     GameLayer(Game* game);
18     void init() override;
19     void processControls() override;
20     void update() override;
21     void draw() override;
22     void keysToControls(SDL_Event event);
23
24     Audio* audioBackground;
25     Text* textPoints;
26     Text* textVida;
27     int points;
28     int newEnemyTime = 0;
29     Player* player;
30     Player* player1;
31     Player* player2; // nuevo jugador
32     Text* textVida2; // mostrar vidas del segundo jugador
33
34     Background* background;
35     Actor* backgroundPoints;
36     list<EnemyPadre> enemies;
37     list<Projectile> projectiles;
38     list<EnemyProjectile> enemyProjectiles;
39
40     bool controlShoot = false;
41     int controlMoveY = 0;
42     int controlMoveX = 0;
43     int controlMoveX2 = 0;
44     int controlMoveY2 = 0;
45     bool controlShoot2 = false;
46
47
48
49
50
51

```

- **Ampliación 5: Cambio de Nave**

- Se añadieron dos naves disponibles para el jugador.
- Al pulsar la tecla 1 o 2, se cambia la nave controlada por el jugador.
- Cada nave tiene imagen, tamaño, velocidad de movimiento y cadencia de disparo diferente.
- Esto permite variar la jugabilidad según la nave seleccionada.

```
void GameLayer::keysToControls(SDL_Event event) {
    int code = event.key.keysym.sym;

    if (event.type == SDL_KEYDOWN) {
        switch (code) {
            | // --- Jugador 1 ---
            case SDLK_1: player1->setShipType(1); break;
            case SDLK_2: player1->setShipType(2); break;
            case SDLK_w: controlMoveY = -1; break;
            case SDLK_s: controlMoveY = 1; break;
            case SDLK_a: controlMoveX = -1; break;
            case SDLK_d: controlMoveX = 1; break;
            case SDLK_SPACE: controlShoot = true; break;
            |
            | // --- Jugador 2 ---
            case SDLK_UP: controlMoveY2 = -1; break;
            case SDLK_DOWN: controlMoveY2 = 1; break;
            case SDLK_LEFT: controlMoveX2 = -1; break;
            case SDLK_RIGHT: controlMoveX2 = 1; break;
            case SDLK_RETURN: controlShoot2 = true; break;
            |
            case SDLK_ESCAPE: game->loopActive = false; break;
        }
    }
}
```

- **Ampliación 6: Multijugador Local**

- Permite que dos jugadores controlen naves simultáneamente en la misma pantalla.
- Cada jugador tiene sus propias teclas de control para moverse y disparar.
- Cada nave tiene vidas independientes y dispara proyectiles de forma separada.
- Mantiene la mecánica de colisiones y puntuación por jugador.

```
NavesL4 GameLayer
10 #include "Text.h"
11 #include "Audio.h"
12 #include <list>
13
14 class GameLayer : public Layer
15 {
16 public:
17     GameLayer(Game* game);
18     void init() override;
19     void processControls() override;
20     void update() override;
21     void draw() override;
22     void keysToControls(SDL_Event event);
23
24     Audio* audioBackground;
25     Text* textPoints;
26     Text* textVida;
27     int points;
28     int newEnemyTime = 0;
29     Player* player;
30     Player* player1;
31     Player* player2; // nuevo jugador
32     Text* textVida2; // mostrar vidas del segundo jugador
33
34     Background* background;
35     Actor* backgroundPoints;
36     list<EnemyPadre*> enemies;
37     list<Projectile*> projectiles;
38     list<EnemyProjectile*> enemyProjectiles;
39
40     bool controlShoot = false;
41     int controlMoveY = 0;
42     int controlMoveX = 0;
43     int controlMoveX2 = 0;
44     int controlMoveY2 = 0;
45     bool controlMove2 = false;
46
47
48 },
49
50
51
```

```
void GameLayer::init()
{
    audioBackground = new Audio("res/musica_ambiente.mp3", true);
    audioBackground->play();

    points = 0;
    textPoints = new Text("0", WIDTH * 0.92, HEIGHT * 0.04, game);

    // --- Jugadores ---
    player1 = new Player(50, 70, game);
    textVida = new Text(std::to_string(player1->vidas), 50, 20, game);

    player2 = new Player(50, 200, game); // posición inicial jugador 2
    textVida2 = new Text(std::to_string(player2->vidas), 700, 20, game);

    background = new Background("res/fondo.png", WIDTH * 0.5, HEIGHT * 0.5, -1, game);
    backgroundPoints = new Actor("res/icono_puntos.png", WIDTH * 0.85, HEIGHT * 0.05, 24, 24, game);

    enemies.clear();
    projectiles.clear();

    enemies.push_back(new EnemyA(300, 50, game));
    enemies.push_back(new EnemyB(300, 200, game));

    // Inicializar controles jugador 2
    controlMoveX2 = 0;
    controlMoveY2 = 0;
    controlShoot2 = false;
}
```

FIN