**Manual técnico**

Juan José Fernández Aristizábal

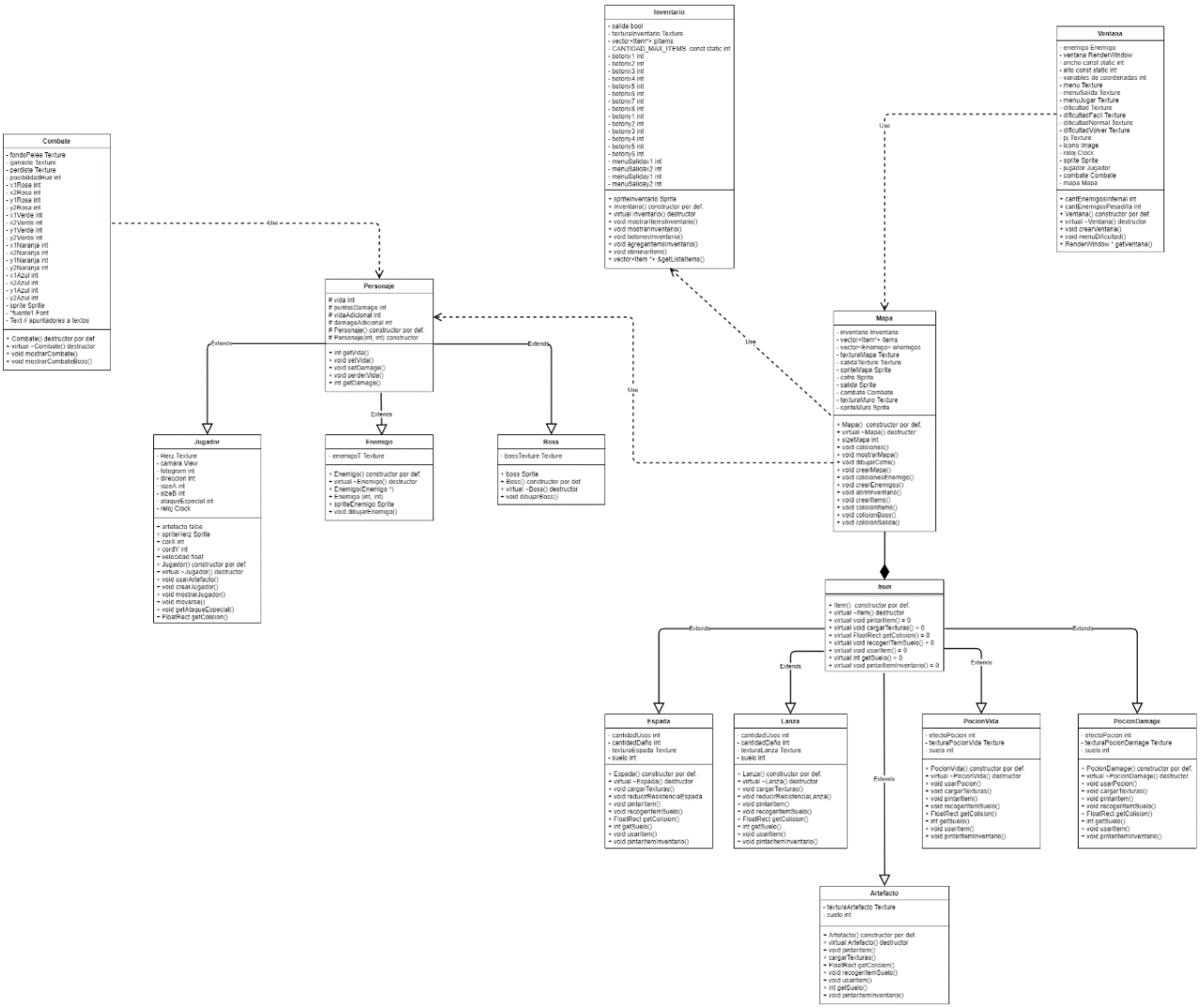
Jean Paul Gonzalez Pedraza

Cristian Camilo Tabares Pérez

Ingeniería de Sistemas y Computación

Programación Orientada a Objetos

## Diagrama de clases:



**Principales Funcionalidades:**

- **Moverse con el jugador:** Es una funcionalidad que requería la asignación de letras del teclado y coordinar las coordenadas dependiendo del movimiento del personaje.

```
void Jugador::moverse() {
    if (Keyboard::isKeyPressed(Keyboard::Right) &&
        cordY <= ancho - 48) {
        cordY += velocidad;
        direccion = 96;
        if (reloj.getElapsedTime().asSeconds() > 0.5f) {
            fotograma += 48;
            reloj.restart();
        }
    }
```

- **Mostrar combate:** Es una clase que se implementó texto de los botones en la ventana, es decir no venia en el fondo y se hace uso de un Font para la misma, asignamos un estilo de combate por turnos teniendo así la opción de huir del mismo y abrir su inventario para el uso de pociones.

```
if(Mouse::getPosition(*ventana).x >= x1Rosa &&
   Mouse::getPosition(*ventana).x <= x2Rosa &&
   Mouse::getPosition(*ventana).y >= y1Rosa &&//coordenadas del boton flip flop trueno
   Mouse::getPosition(*ventana).y <= y2Rosa &&
   sprite.getTexture() == &fondoPelea){
    sleep(milliseconds(100));
    if(Mouse::isButtonPressed(Mouse::Left)){ //Boton rosa funcionalidades de ataque flip flop
        danoR=jugador->getDamage();
        danoRecibido2=enemigo->getDamage();
        enemigo->perderVida(jugador->getDamage());//ataque a enemigo
        jugador->perderVida(enemigo->getDamage());//ataque de enemigo
    }
}
```

- **Mapa-Colisiones:** La funcionalidad colisión fue difícil de realizar debido a que por las coordenadas había partes por las que el personaje podía a travesar estas colisiones, pero finalmente se logró agregar las coordenadas exactas y el uso de intersect en SFML.

```cpp
void Mapa::colisiones(Jugador *jugador, Sprite& objeto) {//colisiones en el mapa
    FloatRect colisionObjeto = objeto.getGlobalBounds();
    if (jugador->getColision().intersects(colisionObjeto)) {//jugador colisiona
        //Parte de abajo del objeto
        if (jugador->getColision().top < colisionObjeto.top
            && jugador->getColision().top + jugador->getColision().height < colisionObjeto.top + coli
            && jugador->getColision().left < colisionObjeto.left + colisionObjeto.width
            && jugador->getColision().left + jugador->getColision().width > colisionObjeto.left) {
            jugador->cordY = (jugador->getColision().left);
            jugador->cordX = (colisionObjeto.top - jugador->getColision().height);
        }//Parte de arriba del objeto
```

- **Crear Ventana**: Se encarga de crear la ventana e implementar el menú principal del juego para darle inicio, encargándose de los botones iniciar y salir.

```cpp
while (ventana.isOpen()){
    salida = false;
    sprite.setTexture(menu);
    Event evento{};
    ventana.clear();
    ventana.draw(sprite);//se dibuja en ventana la textura del menu
    if (Keyboard::isKeyPressed(Keyboard::Escape)){//si precionas escape se saldra del juego
        ventana.close();//cierra la ventana
    }
```

**Funcionalidades con sobre escritura:**

- **Ítem:** La clase ítem es utilizada como padre de los ítems del juego tal como lo es espada, hacha y pociones quienes son las clases hijas de la misma, la clase ítem también es una clase abstracta**.**

```cpp
class Item {//clase abstracta
public:
    Item();
    Sprite spriteItem;
    virtual ~Item();//destructor
    virtual void pintarItem(RenderWindow *ventana, int x, int y) = 0; //pintar item en mapa
    virtual void cargarTexturas() = 0;//cargar las texturas
    virtual FloatRect getColision() = 0;
    virtual void recogerItemSuelo() = 0;//recoger item del suelo
    virtual void usarItem(Jugador * jugador) = 0;//utilizar un item y que haga su funcion
    virtual int getSuelo() = 0;
    virtual void pintarItemInventario(RenderWindow *ventana, int x, int y) = 0;//pintar item en el inv
};
```

- **Personaje:** La clase personaje es utilizada como padre de la clase jugador para el manejo de los atributos del personaje como lo es su vida y daño.

```
class Personaje{
protected:
    int vida, puntosDamage, vidaAdicional = 0, damageAdicional = 0;
    Personaje();
    Personaje(int,int);

public:
    int getVida();// obtener vida jugador
    void setVida(int);// enviar vida jugador
    void setDamage(int);// enviar daño jugador
    void perderVida(int); //recibe el daño y el jugador pierde esa vida
    int getDamage(); // obtener daño jugador
};
```