E2. Proyecto Integrador

SEMANA (5)

15 de Junio de 2023

| Equipo 4 :

Rodrigo Emmanuel Sánchez Melendrez | A01642309

Jose Manuel Martinez Morales | A01734279

Materia: Programación orientada a objetos (Gpo 318)

Profesora: Fabiola Uribe Plata

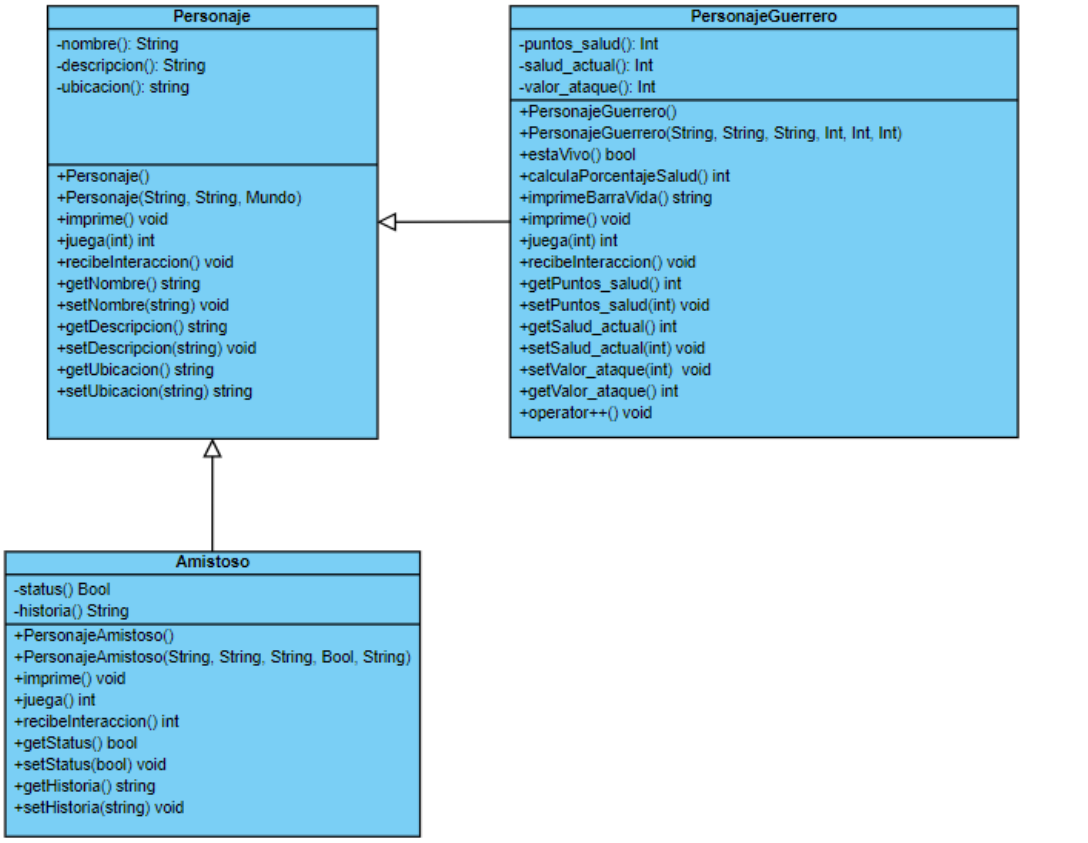
Introducción

El presente proyecto consiste en el desarrollo de un juego de consola basado en texto, donde se aplicarán los conceptos aprendidos en el curso, tales como herencia, polimorfismo, clases abstractas, sobrecarga de operadores y manejo de excepciones. El juego se desarrolla en un entorno compuesto por varios cuartos o mundos, con la presencia de un jugador, personajes amistosos y personajes antagónicos.

El objetivo del jugador es explorar los diferentes cuartos, interactuar con los personajes y enfrentarse a los enemigos para avanzar al siguiente mundo. Además el juego se desarrolla en un flujo sucesivo de exploración de los mundos y enfrentamientos con los personajes antagónicos. Si el jugador pierde contra un enemigo, se mostrará el mensaje correspondiente y se finalizará el juego.

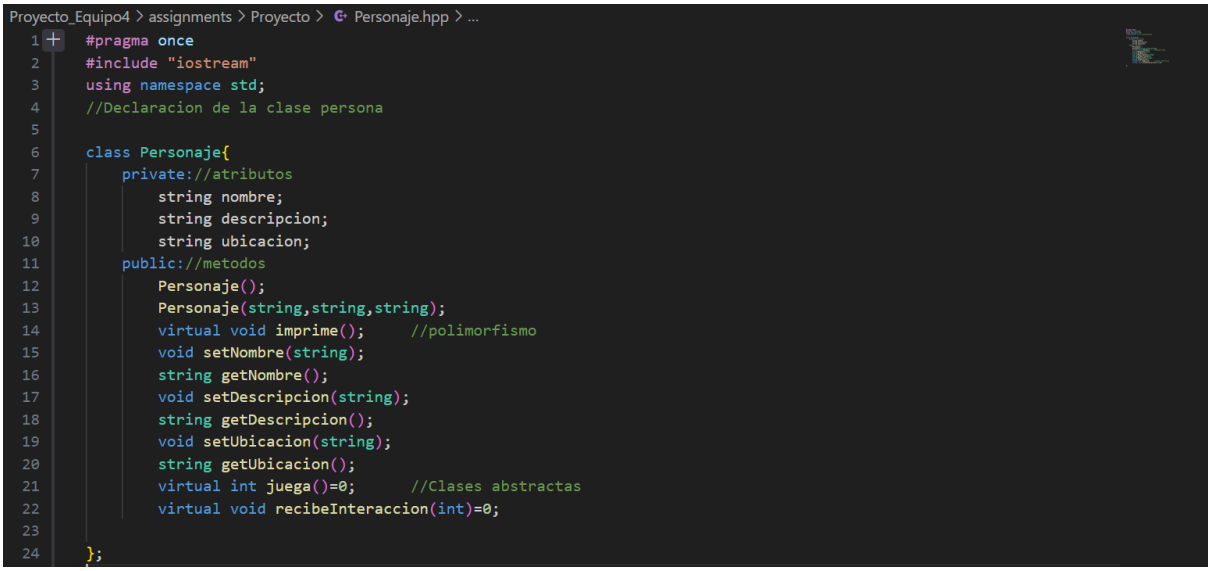
Diagrama UML

Jose Manuel Martinez Morales A01734279
Rodrigo Emmanuel Sánchez Melendrez A01642309



Desarrollo

Declaración de nuestra clase base Personaje:



En esta declaración de la clase principal “Personaje” utilizamos los conceptos vistos en clase, **clases abstractas y polimorfismo**.
Hablando de polimorfismo en este caso el método "imprime()" es declarado como virtual en la clase base "Personaje". Esto permite que las clases derivadas de "Personaje" puedan sobrescribir este método y proporcionar su propia implementación. El polimorfismo se implementa o sucede cuando se llama al método "imprime()" en un objeto de la clase base, pero se ejecuta la implementación específica de la clase derivada.

Y esta a su vez también implementamos el tema de clase abstracta, ya que la clase contiene dos métodos puramente virtuales: "juega()" y "recibeInteraccion(int)". Estos métodos no tienen una implementación definida en la clase base, lo cual indica que la clase "Personaje" es una clase abstracta. Al declarar estos métodos como virtuales puros "= 0", se obliga a las clases derivadas a proporcionar una implementación para ellos. Cualquier clase derivada de "Personaje" debe definir los métodos "juega()" y "recibeInteraccion(int)" antes de poder ser instanciada. Al hacer que la clase base sea abstracta, se establece una estructura común y se garantiza que todas las clases derivadas tendrán una funcionalidad específica.

Declaración de los getters, setters y metodos de Personaje.cpp

Proyecto_Equipo4 > assignments > Proyecto > Personaje.cpp > ...

```cpp
#include "Personaje.hpp"
//Constructores
Personaje::Personaje(){
    nombre="";
    descripcion="";
    ubicacion="";
}

Personaje::Personaje(string _nombre, string _descripcion, string _ubicacion){
    nombre=_nombre;
    descripcion=_descripcion;
    ubicacion=_ubicacion;
}
//setters y getters
void Personaje::setNombre(string _nombre){
    _nombre=nombre;
}

void Personaje::setUbicacion(string _ubicacion){
    _ubicacion=ubicacion;
}

void Personaje::setDescripcion(string _descripcion){
    _descripcion=descripcion;
}
```

```cpp
string Personaje::getNombre(){
    return nombre;
}

string Personaje::getDescripcion(){
    return descripcion;
}

string Personaje::getUbicacion(){
    return ubicacion;
}
//imprime
void Personaje::imprime(){
    cout<<"El nombre es"<<getNombre()<<endl;
    cout<<"Su ubicacion es"<<getUbicacion()<<endl;
}
```

Aquí implementamos los métodos setters, getters y las funciones respectivas de la clase base, usando el polimorfismo.

Declaración de nuestra clase Guerrero hpp:

Proyecto_Equipo4 > assignments > Proyecto > Guerrero.hpp > ...

```cpp
#include "Personaje.hpp"

class Guerrero: public Personaje{//herencia
    private:
        int puntos_salud;
        int salud_actual;
        int valor_ataque;
    public:
        Guerrero();
        Guerrero(string, string, string, int,int,int);
        void setPuntos_salud(int);
        int getPuntos_salud();
        void setSalud_actual(int);
        int getSalud_actual();
        void setValor_ataque(int);
        int getValor_ataque();
        void imprime();         //polimorfismo
        int juega();            //usamos las clases abstractas de personaje
        void recibeInteraccion(int);
        bool estaVivo();
        int calculaPorcentajeSalud();
        void imprimeBarraVida();
        void operator++();//sobrecarga de operador

};
```

En esta clase “Guerrero” usamos herencia, porque la clase "Guerrero" hereda de la clase base "Personaje" mediante la declaración de "public Personaje". Esto

establece una relación de "es-un" entre "Guerrero" y "Personaje". Un guerrero es un tipo de personaje. Al heredar de la clase "Personaje", la clase "Guerrero" adquiere todas las características y comportamientos definidos en la clase base. Esto incluye los atributos y métodos de la clase "Personaje", como nombre, descripción, ubicación, entre otros, en este caso la herencia permite reutilizar y extender el código de la clase base.

También se implementa el polimorfismo aquí, ya que la clase "Guerrero" redefine el método "imprime()" heredado de la clase base "Personaje". Al declararse este método como "virtual" en la clase base y volverlo a declarar en la clase derivada, se establece el polimorfismo. Cuando se llama al método "imprime()" en un objeto de la clase "Guerrero", se ejecuta la implementación específica de "imprime()" en la clase "Guerrero", en lugar de la implementación heredada de la clase base. Esto permite que diferentes tipos de personajes tengan comportamientos de impresión distintos, pero se utilice una interfaz común.

Declaración del Guerrero cpp:

```
52      if (porcentaje <= 0) {
53          cout << "\033[31m" << string(20, '=') << "\033[0m" << endl;
54      } else {
55          cout << string(porcentaje, '%') <<string(porcentaje, '%') << string(20 - porcentaje, '=') << endl;
56      }
57  }
58
59  int Guerrero::juega(){
60      int rando=(rand()%getValor_ataque());
61      cout<<getNombre()<<" ataca con "<<rando<<" de dano"<<endl;
62      return rando;
63  }
64
65  void Guerrero::recibeInteraccion(int dano){                //utilizamos las clases abstractas de personaje
66      salud_actual-=dano;
67      if(salud_actual<0){
68          cout<<getNombre()<<" ha muerto" <<endl;
69      }
70      else{
71          cout<<"Vida restante "<<getSalud_actual()<<endl;
72          imprimeBarraVida();
73      }
74  }
```

```
76  void Guerrero::imprime(){
77      Persona:imprime();
78      cout<<"La barra de vida es:"<<endl;
79      imprimeBarraVida();
80  }
81
82  void Guerrero::operator++(){        //sobrecarga de operador
83      valor_ataque+=10;
84  }
```

En este fragmento de código usamos también la sobrecarga de operadores como función, porque se utiliza la sobrecarga del operador "++" en la clase "Guerrero" mediante la definición del método "operator++()", la sobrecarga del operador "++" permite incrementar el valor de "valor_ataque" en 10 unidades cuando se utiliza el operador en un objeto de la clase "Guerrero". Esto proporciona una forma conveniente de aumentar el valor de ataque del guerrero sin necesidad de llamar a un método adicional.

Y al igual con la clase abstracta, al heredar de la clase abstracta, la clase "Guerrero" se compromete a implementar los métodos virtuales puros definidos en la clase base. Esto asegura que cada tipo de personaje derivado de "Personaje" tenga una implementación adecuada para los métodos de juego y la interacción.

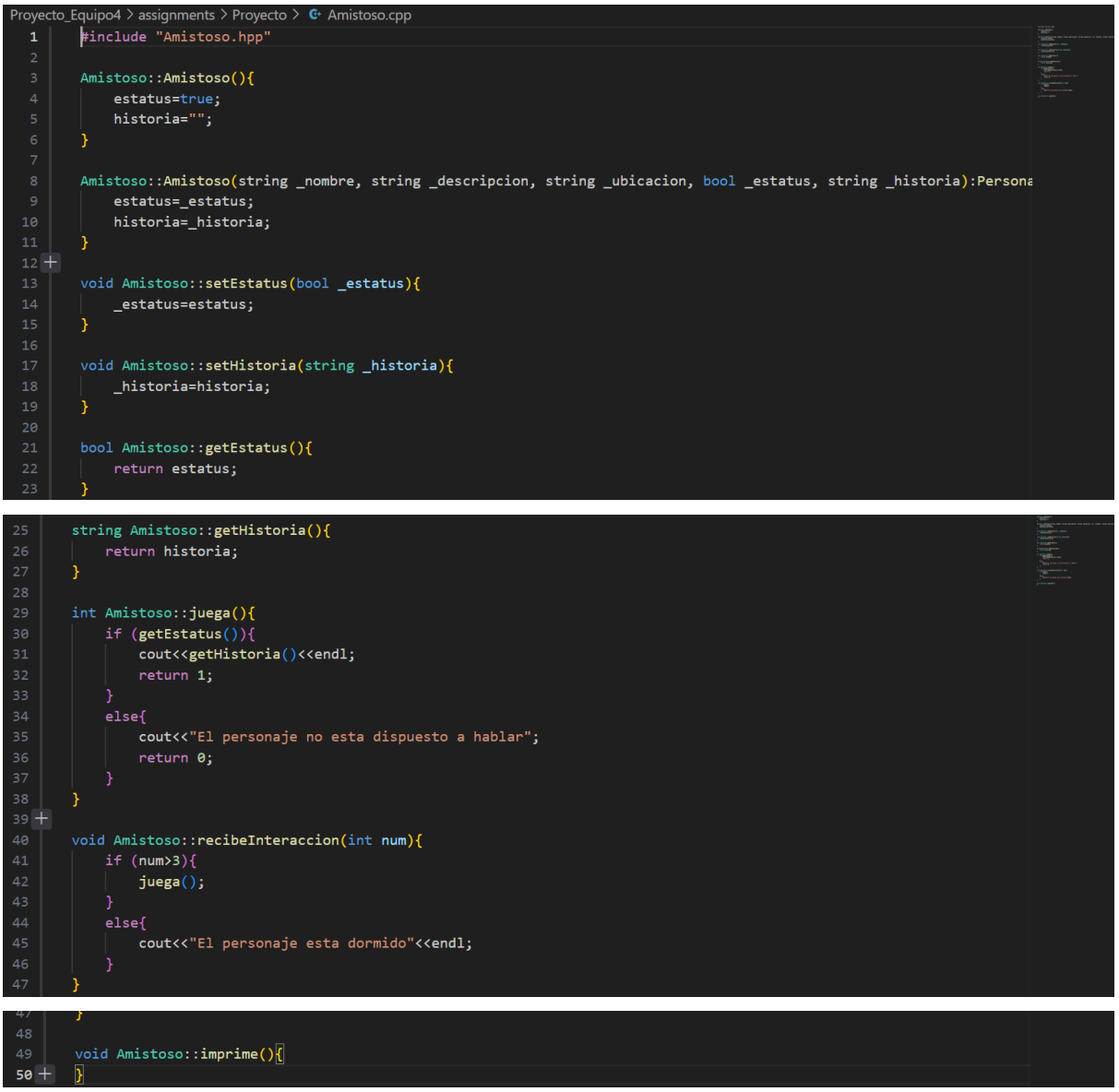
Declaración de la clase Amistoso hpp:

Proyecto_Equipo4 > assignments > Proyecto > Amistoso.hpp > ...

```
1   #include "Personaje.hpp"
2   using namespace std;
3
4   class Amistoso:public Personaje{        //herencia
5       private:
6           bool estatus;
7           string historia;
8       public:
9           Amistoso();
10          Amistoso(string, string, string, bool, string);
11          int juega();                    //utilizamos las clases abstractas de Personaje
12          void recibeInteraccion(int);
13          void setEstatus(bool);
14          void setHistoria(string);
15          bool getEstatus();
16          string getHistoria();
17          void imprime();     //polimorfismo
18
19  };
20
```

Al igual que en Guerrero, en la clase “Amistoso”, utilizamos herencia, ya que es una clase derivada de “Personaje” y también implementamos el polimorfismo y clases abstractas.

Declaración del Amistoso cpp:



```
Proyecto_Equipo4 > assignments > Proyecto > Amistoso.cpp
#include "Amistoso.hpp"

Amistoso::Amistoso(){
    estatus=true;
    historia="";
}

Amistoso::Amistoso(string _nombre, string _descripcion, string _ubicacion, bool _estatus, string _historia):Persona
    estatus=_estatus;
    historia=_historia;
}

void Amistoso::setEstatus(bool _estatus){
    _estatus=estatus;
}

void Amistoso::setHistoria(string _historia){
    _historia=historia;
}

bool Amistoso::getEstatus(){
    return estatus;
}
```



```
string Amistoso::getHistoria(){
    return historia;
}

int Amistoso::juega(){
    if (getEstatus()){
        cout<<getHistoria()<<endl;
        return 1;
    }
    else{
        cout<<"El personaje no esta dispuesto a hablar";
        return 0;
    }
}

void Amistoso::recibeInteraccion(int num){
    if (num>3){
        juega();
    }
    else{
        cout<<"El personaje esta dormido"<<endl;
    }
}
```



```
}

void Amistoso::imprime(){
}
```

En este código al igual que el Guerrero cpp, implementamos el uso de polimorfismo, al usar el “método imprime()” de la clase base “Personaje”, al igual que el uso de la clase abstracta.

Declaración del excersice cpp (Main):

Proyecto_Equipo4 > assignments > Proyecto > exercise.cpp > ...

```cpp
#include <iostream>
#include <vector>
#include <unistd.h>
#include "Guerrero.hpp"
#include "Amistoso.hpp"
//funcion para el menu
void mostrarMenu() {
    cout << "---------- MENÚ ----------" << endl;
    cout << "1. Jugar" << endl;
    cout << "2. Salir" << endl;
    cout << "----------------------------" << endl;
    cout << "Ingrese su opción: ";
}
//funcion para que los personajes guerrero juegen
void fight(Guerrero* jugador, Guerrero* malo){
    int damagej;
    int damagem;
    cout<<malo->getNombre()<<" se entrometio en tu camino"<<endl;
    while (jugador->estaVivo() and malo->estaVivo()){
        damagej=jugador->juega();
         cout.flush();                    //Utilizamos flush y sleep para que el usuario pueda leer el texto y el
 sleep(1);
        malo->recibeInteraccion(damagej);
        malo->imprimeBarraVida();
        if (malo->estaVivo()){
```

```cpp
cout.flush();
 sleep(1);
            damagem=malo->juega();
            jugador->recibeInteraccion(damagem);
            jugador->imprimeBarraVida();
        }
            else{
                cout.flush();
 sleep(1);
                cout<<"Has vencido a "<<malo->getNombre()<<endl;
                break;
        };
    };
}
//funcion para que Guerrero y Amistoso jueguen
void meeting(Guerrero* jugador, Amistoso* bueno){
   cout.flush();
 sleep(1);
   cout<<"Te encuentras con "<<bueno->getNombre()<<endl;
   int random=(rand()%10);
   bueno->recibeInteraccion(random);
   if(random>3){
       cout<<"Toma esto es la espada del heroe te aumentara el ataque"<<endl;
       cout.flush();
 sleep(1);
```

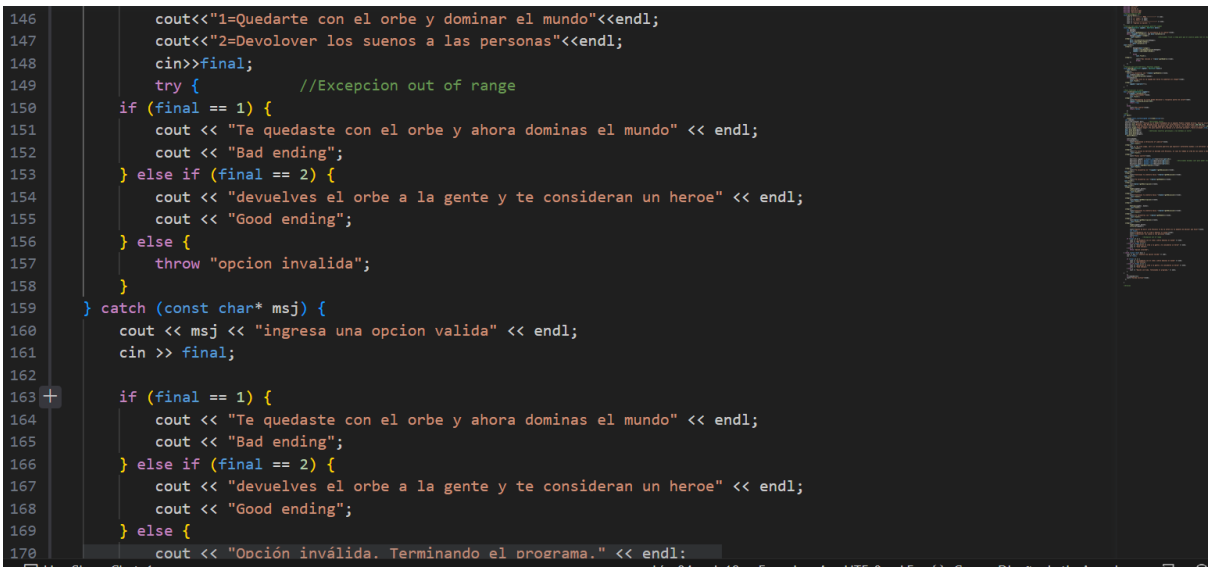
```cpp
            jugador->operator++();
    }
}

//Para finalizar la pelea
bool victoria(Guerrero* jugador){
    if (jugador->estaVivo()){
        cout<<"Felicidades"<<endl;
        cout.flush();
 sleep(1);
        cout<<"Encuentras un villa donde descansar y recuperas puntos de salud"<<endl;
        jugador->setSalud_actual(100);
        return true;
    }
    else{
        cout<<"Has muerto"<<endl;
        return false;
    };
    }
```

```cpp
int main()
{
    srand(static_cast<unsigned int>(time(nullptr)));
  int comando;
  vector<Personaje*> pers;      //Utilizamos vector
  Guerrero ely("Elysian","Elysian es un hombre bondadoso de un pequeno Pueblo llamado Aurelia, Elysian se prepara i
  Guerrero mor("Morgana","Una Malvada bruja al servicio de Lord Draconis","Torre Destruida",100,100,10);
  Guerrero lord("Lord Draconis","El malvado emperador que robo el orbe de los suenos","Castillo infernal",100,100,1
  Amistoso migan("Miguel Angel","Un gran maestro de la katana y el chiflido aullador","Valle olvidado",true,"Oh vie
  pers.push_back(&ely);
  pers.push_back(&mor);         //definimos nuestros personajes y los metemos al vector
  pers.push_back(&lord);
  pers.push_back(&migan);
    mostrarMenu();

    cin>>comando;
    if(comando==1){
        cout<<"Bienvenido a Chronicles of Lumaria"<<endl;
         cout.flush();
  sleep(3);
        cout << "En este juego, serás un valiente guerrero que explorará diferentes mundos y se enfrentará a persor
         cout.flush();
  sleep(3);
        cout<<"Tu mision es derrotar al malvado Lord Draconis, el cual ha robado el orbe de los suenos y ahora las
         cout.flush();
```

```cpp
   sleep(3);
         cout<<"Buena suerte"<<endl;

         Guerrero* jugador = dynamic_cast<Guerrero*>(pers[0]);
         Guerrero* malo1 = dynamic_cast<Guerrero*>(pers[1]);         //Utilizamos dinamyc cast para poder utilizar l
         Guerrero* malo2 = dynamic_cast<Guerrero*>(pers[2]);
         Amistoso* bueno = dynamic_cast<Amistoso*>(pers[3]);
         cout<<jugador->getDescripcion()<<endl;
          cout.flush();
   sleep(2);
         cout<<"Te encuentras en "<<jugador->getUbicacion()<<endl;
  cout.flush();
   sleep(2);
         cout<<"Continuas tu aventura hacia "<<malo1->getUbicacion()<<endl;
  cout.flush();
   sleep(2);
         cout<<"Te encuentras con "<<malo1->getNombre()<<endl;
  cout.flush();
   sleep(2);
         cout<<malo1->getDescripcion()<<endl;
  cout.flush();
   sleep(2);
         fight(jugador,malo1);
         victoria(jugador);
          cout.flush();
```

```cpp
   sleep(2);
         cout<<"Continuas tu aventura hacia "<<bueno->getUbicacion()<<endl;
          cout.flush();
   sleep(2);
         cout<<bueno->getDescripcion()<<endl;
          cout.flush();
   sleep(2);

         meeting(jugador, bueno);
          cout.flush();
   sleep(2);
         cout<<"Continuas tu aventura hacia "<<malo2->getUbicacion()<<endl;
          cout.flush();
   sleep(2);
         cout<<"Te encuentras con "<<malo2->getNombre()<<endl;
          cout.flush();
   sleep(2);
         cout<<malo2->getDescripcion()<<endl;
          cout.flush();
   sleep(2);
         fight(jugador,malo2);
         victoria(jugador);

         cout<<"Apunto de morir Lord Draconis te da la esfera es tu momento de decidir que hacer"<<endl;
         int final;
```

Cargando...



```
146         cout<<"1=Quedarte con el orbe y dominar el mundo"<<endl;
147         cout<<"2=Devolover los suenos a las personas"<<endl;
148         cin>>final;
149         try {           //Excepcion out of range
150     if (final == 1) {
151         cout << "Te quedaste con el orbe y ahora dominas el mundo" << endl;
152         cout << "Bad ending";
153     } else if (final == 2) {
154         cout << "devuelves el orbe a la gente y te consideran un heroe" << endl;
155         cout << "Good ending";
156     } else {
157         throw "opcion invalida";
158     }
159 } catch (const char* msj) {
160     cout << msj << "ingresa una opcion valida" << endl;
161     cin >> final;
162 
163     if (final == 1) {
164         cout << "Te quedaste con el orbe y ahora dominas el mundo" << endl;
165         cout << "Bad ending";
166     } else if (final == 2) {
167         cout << "devuelves el orbe a la gente y te consideran un heroe" << endl;
168         cout << "Good ending";
169     } else {
170         cout << "Opción inválida. Terminando el programa." << endl;
```

```
171     }
172 }
173     };
174     if(comando==2){
175     cout<<"Salida exitosa"<<endl;
176   };
177 
178 }
179 
180 //Gracias
```

En este desarrollo de código del Main, usamos de todo lo visto en clase y que anteriormente ya mencionamos, iniciando ahora por excepciones, Se utiliza una excepción para manejar la situación en la que el usuario ingresa una opción inválida en el momento de decidir el final del juego. La excepción "throw" se utiliza para lanzar un mensaje de error indicando que la opción ingresada es inválida, el bloque "try-catch" se utiliza para capturar la excepción y proporcionar una respuesta adecuada, esto permite controlar y manejar los casos en los que se ingresan opciones no válidas, evitando que el programa se bloquee o se comporte de manera inesperada.

También usamos el polimorfismo en este código, se utiliza mediante los punteros a la clase base "Personaje" para almacenar objetos de las clases derivadas "Guerrero" y "Amistoso", lo que nos ppermite tratar a los objetos de diferentes clases derivadas como si fueran objetos de la clase base, lo que facilita el procesamiento y la manipulación de múltiples personajes en una estructura de datos común. Por ejemplo, se utiliza polimorfismo al llamar a los métodos "juega()", "recibeInteraccion()" y "imprimeBarraVida()" en los objetos de las clases "Guerrero" y "Amistoso" a través de punteros a la clase base "Personaje", así promueve la flexibilidad y extensibilidad del código, ya que se pueden agregar fácilmente nuevas clases derivadas de "Personaje" sin modificar el código existente.

En resumen, se utilizan excepciones para manejar casos de entrada inválida, polimorfismo para tratar objetos de diferentes clases como si fueran de la misma clase base, y una clase abstracta para definir una interfaz común y establecer una

base para el polimorfismo en el manejo de personajes en el juego.

Conclusiones

Rodrigo Emmanuel:

Puedo decir que me llevo mucho aprendizaje de POO (programación orientada a objetos), el saber cómo perfeccionar los códigos que implementamos, para la resolución de problemas de la vida diaria, y saber que hoy en día existen estas herramientas tan poderosas como lo es la tecnología y el uso de software, también como se hace la reutilización de código, para hacer más entendibles a la hora de trabajar, es por ello, que aprendí mucho, y en complemento con lo antes ya aprendido de POO, profundice más en los conocimientos y aprendizajes de temas, que vimos a lo largo de este curso y que usamos en este proyecto.

Concluyendo puedo decir que, la programación orientada a objetos es un enfoque poderoso y flexible para el desarrollo de software, que se basa en la creación y manipulación de objetos y proporcionan las herramientas necesarias para modelar y resolver problemas complejos de manera modular, reutilizable y fácil de entender.

Jose Manuel:

Gracias a la programación orientada a objetos puede adquirir una base sólida para el desarrollo de software de calidad. POO permite adaptarnos a los cambios y requisitos de un proyecto,

En conclusión, la POO es una herramienta fundamental en el desarrollo de software moderno y en la carrera de ingeniería en tecnologías computacionales. A través de su los recursos de la materia, se puede abordar de manera eficiente el problema del reto. De igual manera POO les permite a los programadores construir eficientes en el mundo tecnológico en que evoluciona de manera exponencial cada dia.

Referencias bibliográficas

FRA2. (s. f.). http://www.lcc.uma.es/~eat/services/poo_c/fra2.html

TylerMSFT. (2023, 3 abril). *Instrucciones try, throw y catch (C++)*. Microsoft Learn. https://learn.microsoft.com/es-es/cpp/cpp/try-throw-and-catch-statements-cpp?view=msvc-170

Davidson, S. R. (s. f.). *C++ con Clase | Librería ANSI C (stdio/fflush)*. © 2000 Steven R. Davidson. https://conclase.net/c/librerias/stdio/fflush#:~:text=Funci%C3%B3n%20fflush%20ANSI%20C,el%20comportamiento%20no%20est%C3%A1%20definido.

González, J. D. M. (2020). Librerías o Bibliotecas. *www.programarya.com*. https://www.programarya.com/Cursos/C++/Bibliotecas-o-Librerias

GitHub Desktop. (s. f.). GitHub Desktop. https://desktop.github.com/