JOSE RAMÓN GASCON ENRIQUEZ

Descripción breve

Diseñar e implementar un programa que simule un juego de roles (RPG) donde los jugadores puedan crear equipos de héroes y enfrentarse a enemigos.

INFORME de c/c++

**Introducción:**

En este mundo de fantasía, los jugadores podrán crear diferentes tipos de personajes, como guerreros y hechiceros, cada uno con habilidades únicas. Además, podrán enfrentarse entre sí en emocionantes batallas utilizando estrategias y tácticas para lograr la victoria.

A lo largo de este informe, se describirá detalladamente el diseño y la implementación del programa, incluyendo las clases y estructuras utilizadas, la lógica de juego y las funcionalidades principales. También se proporcionarán ejemplos prácticos para ilustrar el funcionamiento del programa.

Este proyecto representa una oportunidad emocionante para explorar el mundo de la programación de juegos y combinar la creatividad y la lógica para crear una experiencia de juego envolvente y entretenida.

**Objetivo:**

Este proyecto tiene como objetivo crear un programa en C++ que simule un juego de roles (RPG) donde los jugadores pueden formar equipos de héroes y enfrentarse a desafiantes enemigos. Se emplearán los principios de la programación orientada a objetos (POO) para modelar las diferentes entidades del juego, como personajes, equipos y la lógica de batalla.

FUNCIONES DEL PROGRAMA

Estas funciones proporcionan una experiencia de juego de roles permitiendo a los jugadores enfrentarse en emocionantes batallas

1. **Batalla:**

La clase `Batalla` contiene métodos principales que coordinan la lógica de una batalla entre dos equipos de jugadores:

void Batalla::mostrarTeam(){  
  
 cout<<"\t Arena "<<'\n';  
 cout<<equipo1.getNombre()<<'\t'<<'\t'<<'\t'<<equipo2.getNombre()<<'\n';  
 int i = 0;  
 while (i < 3){  
 if (equipo1.size()>i){  
 cout<<(i+1)<<" "<<equipo1.getListPersonajes()[i]->getNombre()<<" HP "<<equipo1.getListPersonajes()[i]->getHp()<<'\t'<<'\t';  
 if (equipo2.size()>i){  
 cout<<(i+1)<<" "<<equipo2.getListPersonajes()[i]->getNombre()<<" HP "<<equipo2.getListPersonajes()[i]->getHp()<<'\n';  
 }else cout<<'\n';  
 } else if(equipo2.size()>i)  
 cout<<(i+1)<<" "<<equipo2.getListPersonajes()[i]->getNombre()<<" HP "<<equipo2.getListPersonajes()[i]->getHp()<<'\n';  
 i++;  
 }  
  
}

Muestra la información de los equipos participantes, incluyendo los nombres de los equipos y los nombres y puntos de vida de los personajes.

void Batalla::turno\_Jugador1() {  
  
 int i = 0;  
 while(i<equipo1.size()){  
 mostrarTeam();  
 cout<<"Turno de: "<<equipo1.getListPersonajes()[i]->getNombre()<<'\n';  
 int resividorId;  
 cout<<"Introduce el id del Heroe que vas a atacar: "<<'\n';  
 cin>>resividorId;  
 if (resividorId <= 0 or resividorId > equipo2.size()){  
 cout<<"Ingrese un id valido"<<'\n';  
 continue;  
 }if (Funciones().atacar(equipo1.getListPersonajes()[i],equipo2.getListPersonajes()[resividorId-1])){

caidos.addPersonaje(equipo2.getListPersonajes()[resividorId-1]);  
 string\* nombreP = const\_cast<string \*>(&equipo2.getListPersonajes()[resividorId-1]->getNombre());  
 equipo2.removePersonaje(nombreP);  
 if (equipo2.getListPersonajes().empty()) break;  
 }  
 i++;  
 }}

Simula el turno del jugador 1 para atacar al equipo contrario, solicitando al jugador que seleccione un personaje para atacar y realizando el ataque.

\* void Batalla::turno\_Jugador2() {  
  
 int i = 0;  
 while(i<equipo2.size()){  
 cout<<"Turno de: "<<equipo2.getListPersonajes()[i]->getNombre()<<'\n';  
 random\_device rd;  
 mt19937 gen(rd());  
 uniform\_int\_distribution<int> distribucion(1, 3);  
 int resividorId = distribucion(gen);  
 if (Funciones().atacar(equipo2.getListPersonajes()[i],equipo1.getListPersonajes()[resividorId-1])){  
 caidos.addPersonaje(equipo1.getListPersonajes()[resividorId-1]);  
 string\* nombreP = const\_cast<string \*>(&equipo1.getListPersonajes()[resividorId-1]->getNombre());  
 equipo1.removePersonaje(nombreP);  
 if (equipo1.getListPersonajes().empty()) break;  
 }  
 i++;  
 }  
  
}

Similar a `turno\_Jugador1()`, pero simula el turno del jugador 2 (o de la computadora), donde los ataques son realizados de forma aleatoria.

void Batalla::Mostar\_Arena() {  
  
 while (equipo1.size()>0 && equipo2.size()>0){  
 turno\_Jugador1();  
 if (equipo2.getListPersonajes().empty()){  
 break;  
 }  
 cout<<"Turno del otro Jugador:"<<'\n';  
 turno\_Jugador2();  
 }  
 if (equipo2.getListPersonajes().empty()){  
 ganador=equipo1;  
 cout<<"Felicidades tu equipo ha resultado ganador, bien echo"<<'\n'<<ganador.getNombre()<<'\n';  
 ganador.mostrarEquipo();  
 } else {  
 ganador=equipo2;  
 cout<<"Derrota, mejor suerte la proxima"<<'\n'<<ganador.getNombre()<<'\n';  
 ganador.mostrarEquipo();  
 }  
 cout<<"Heroes caidos"<<'\n';  
 caidos.mostrarEquipo();  
  
}

Coordina el flujo completo de la batalla, alternando entre los turnos de los jugadores hasta que uno de los equipos queda sin personajes. Determina el ganador de la batalla y muestra el resultado.

Estos métodos constituyen la estructura principal para simular una batalla en el juego.

1. **Equipo:**

Aquí tienes una lista de los métodos importantes de la clase `Equipo` junto con una breve explicación de cada uno:

void Equipo::setListPersonajes(const vector<Personaje \*> &listPersonajes) {  
 Equipo::listPersonajes = listPersonajes;  
}

- Permite establecer la lista de personajes del equipo con una lista proporcionada.

- Facilita la actualización o modificación de la lista de personajes del equipo.

//Metodo con el operador = sobrecargado  
Equipo Equipo::operator=(Equipo e2) {  
  
 nombre = e2.nombre;  
 listPersonajes = e2.listPersonajes;  
 return \*this;  
  
}

- Sobrecarga el operador de asignación para copiar el contenido de otro equipo en el equipo actual.

- Permite una gestión eficiente de los equipos durante el juego, especialmente en situaciones donde se necesite duplicar un equipo existente.

Estos métodos son esenciales para gestionar la información y las operaciones relacionadas con los equipos de personajes en el juego. Permiten una interacción efectiva con los equipos y optimizan la experiencia de juego para los jugadores.

La clase `Equipo` facilita la gestión de equipos de personajes en el juego. Permite acceder y manipular la lista de personajes del equipo, proporciona identificación clara del equipo mediante su nombre, muestra detalles de los personajes y permite copiar equipos con el operador de asignación. En resumen, la clase `Equipo` brinda una plataforma estructurada para organizar y gestionar personajes, optimizando la experiencia de juego.

3. **Personaje**

1.Constructor y Destructor:

//Contructor

Personaje::Personaje(const string &nombre,int hp) : nombre(nombre), nivel(1), hp(hp) {}  
//Destructor  
Personaje::~Personaje() {}

- La clase `Personaje` tiene un constructor para inicializar un personaje con un nombre y nivel dados, y un destructor para liberar cualquier recurso asignado dinámicamente cuando el objeto se destruye.

2. Herencia:

- `Guerrero`, `Hechicero` y `GuerreroHechicero` son subclases de `Personaje`, lo que significa que heredan los atributos y métodos de la clase base.

- Esto permite definir diferentes tipos de personajes con habilidades y características específicas, manteniendo la estructura y la funcionalidad común proporcionada por la clase `Personaje`.

En resumen, la clase `Personaje` proporciona la estructura básica para representar a los personajes del juego, con métodos para mostrar información y atributos que definen las características fundamentales de cada personaje. Además, sirve como base para la implementación de diferentes tipos de personajes a través de la herencia en el sistema de clases del juego.

**Conclusiones:**

El sistema de juego de roles desarrollado en C++ representa una amalgama de diseño sólido, modularidad y funcionalidad rica para ofrecer una experiencia de juego inmersiva. A través de la implementación de clases como `Personaje`, `Equipo` y `Batalla`, se logra una estructura robusta que permite a los jugadores crear equipos, participar en batallas estratégicas y experimentar un progreso significativo a lo largo del juego.

La flexibilidad del sistema, respaldada por la programación orientada a objetos, permite una fácil extensibilidad y personalización. La inclusión de subclases como `Guerrero`, `Hechicero` y `GuerreroHechicero` agrega variedad y profundidad al juego, ofreciendo a los jugadores una amplia gama de opciones para construir sus equipos y desarrollar estrategias únicas.

Además, la implementación de funcionalidades como la visualización de información de personajes y la simulación de batallas proporciona una experiencia de juego dinámica y envolvente. Los jugadores pueden sumergirse en un mundo de fantasía y aventura, donde la toma de decisiones estratégicas y el trabajo en equipo son clave para alcanzar la victoria.

En última instancia, el sistema de juego de roles en C++ brinda una plataforma versátil y emocionante para la creación y el disfrute de historias épicas y emocionantes aventuras. Con su diseño sólido y sus amplias posibilidades de expansión, el sistema promete ofrecer horas de entretenimiento y satisfacción a los entusiastas de los juegos de rol.