Tecnológico de Monterrey

Programación Orientada a Objetos

Explicación de mejoras e innovaciones

Joel Guadalupe García Guzmán A01713785

Pedro Oscar Pérez Murueta

15 de junio de 2025

Descripción breve de mejoras:

Agregado elementos de **suerte**, valor que se define con una pregunta sobre un número aleatorio. Este elemento modifica los valores de ataque de los personajes, para los enemigos se genera aleatoriamente.

Valor de **velocidad**, elemento que se define según los valores de vida, escudo y suerte. Ayuda a definir quién atacará primero en el combate.

Mejoras en los datos impresos a lo largo del combate, mostrando la velocidad del héroe y enemigo a lo largo del combate, mostrando quien ataca y cuanta vida le queda a quien recibe el ataque. Comprobación para las peleas que van a durar por la eternidad si ocurre un caso donde ninguno de los 2 personajes hace daño al otro.

Cambios de **balance**, modificar los atributos de *recover()* para todos los personajes, y también algunos otros valores para asegurar que el juego sea retador pero posible de ganar. Cada personaje tiene la oportunidad de ganar el juego de alguna forma u otra.

Cambios en el main

Creado la función *assignRandom()*

```
int assignRandom(){

    try {
        int guess, answer = (rand() % 101) - 1;
        cout << "To determine your heros luck you need to guess a
number"
        << " from 1 to 100" << endl;
        cin >> guess;
        cout << "Nice try, the number was " << answer << endl;
        if (guess<=100 && guess>=0){
            int separation = guess - answer;
            if (separation<0){
                separation = separation * -1;
            }
            return 100-separation;
        }
        else{
            throw guess;
        }
    }
    catch(int guess){
        cout << "Invalid number"<< endl;
        cout << "Your hero has the worst luck possible :(" << endl;
```

```
        return 1;
    }
}
```
Función que pide al usuario adivinar un número generado aleatoriamente entre 1 y 100 para determinar la suerte del héroe.
Usa *try* y *catch* para comprobar si el número ingresado está dentro del rango especificado.

Al momento de usar *createHero()* le agrega la suerte obtenida en el segmento previo.

```
    int gobLuck = (rand() % 100);
    int orcLuck = (rand() % 85)+15;
    int draLuck = (rand() % 70)+30;
```
la suerte de los enemigos es generada aleatoriamente, pero entre más fuerte es el enemigo más suerte va a tener. Para esto hago que la suerte del *Goblin* sea entre 0-100, del *Orc* sea de 15-100 y del *Dragon* sea de 30-100. Esto para hacer las batallas más desafiantes a lo largo del juego.

```
        cout << "The next battle is against " <<
levels[i]->getEnemy()->getName() << endl;
        cout << "Your hero has " << hero->getSpeed() <<" of speed" <<
endl;
        cout << "The enemy has " << levels[i]->getEnemy()->getSpeed()
<<" of speed" << endl << endl;
        cout << "Press any key to continue" << endl;

        cin >> advance;
```
Agregado impresiones del texto informando al usuario de datos como la velocidad de enemigos, que enemigo es el siguiente, y una tecla de confirmación para continuar con la batalla.

Cambios en la clase *Character* y las clases herencia

Creado el atributo *luck* y *speed* y cambiado el atributo *strength* de int a double para que funcione con mi nuevo sistema de ataque.

Sobrecargado al operador < el cual va a comparar la velocidad de 2 *Character* para saber quien va a atacar primero en las rondas.

```
speed = health-mana/shield+0.5*luck;
```
El atributo de velocidad va a depender de otros factores, como se ve en la parte superior. Esta se define en el momento de crear el objeto.

```
void Warrior::attack(Character* a){
    double crit = 1;
    if (mana>=10){
```

```
        crit = crit*2;
        mana = mana-10;
    }

    crit = crit * (luck/70);

    a->takeDamage(strength*crit);
}
```

Esta es la lógica del ataque de una de las clases herencia. Conservé el aspecto de los golpes críticos, pero le agregue el aspecto de la suerte, para cada diferente clase es ligeramente diferente.

También altere el método *recover()* el cual en lugar de restablecer la salud y el maná a su máximo, le aumenta puntos cada vez que se usa, también le resta puntos a su velocidad, para así agregar un elemento de estrategia sobre usarlo o no usarlo.

Tabla de las diferentes clases y enemigos del juego

	Warrior	Archer	Mage
Starting Hp	80	60	65
Starting mana	30	50	100
Strength	40	30	20
Shield	20	15	10
Luck	1-100	1-100	1-100
Speed	health-mana/shield+0.5*luck		
Crit attack	mana > 10 X2 strength + factor de suerte	mana > 30 X3 strength + factor de suerte	mana > 30 X4 strength + factor de suerte
Recover	+30 hp +8 mana -15 velocidad	+25 hp +30 mana -10 velocidad	+30 hp +30 mana -5 velocidad

Tabla con valores relevantes a las diferentes clases de héroes

	Goblin	Orc	Dragon
Starting Hp	20	75	100
Starting mana	0		
Strength	15	45	60
Shield	5	15	30
Luck	1-100	15-100	30-100
Speed	health-mana/shield+0.5*luck		
Recover	+15 hp +15 mana	+15 hp +15 mana	+15 hp +15 mana

Tabla con valores relevantes a los diferentes enemigos

Cambios en la clase *Level*

Todos los cambios de *Level* fueron en el método *execute()*, el resto se quedó con los requerimientos de la actividad

```
    bool continueBattle = true;
    Character* winner;
    int nOfTurns=1;
```

Agregamos estas variables que nos servirán para el funcionamiento de *execute()*

```
    if (!hero->isAlive() &! enemy->isAlive()){
        continueBattle = false;
    }
```

nos aseguramos que los 2 estén vivos

```
while (continueBattle)
```

Usaremos la variable *continueBattle* la cual se actualizará en el momento que alguien muera

```
if (*enemy < *hero) //dentro del main

bool Character::operator<(const Character &other){
    return (this->getSpeed() < other.getSpeed());
} //dentro de Character
```

Comparamos la velocidad del enemigo con la del héroe para ver quien ataca primero

```
hero->attack(enemy);
            cout << hero->getName() << " attacks " << enemy->getName()
<< endl;
```

```
            cout << enemy->getName() << " has " << enemy->getHealth()
<< endl;
            if (!enemy->isAlive()){
                break;
            }
            enemy->attack(hero);
            cout << enemy->getName() << " attacks " << hero->getName()
<< endl;
            cout << hero->getName() << " has " << hero->getHealth() <<
endl;
```

Caso en el que el héroe tiene más velocidad, ataca primero, comprueba que el enemigo sigue vivo, y si sigue vivo seguidamente ataca el enemigo.

En el caso que el enemigo sea más rápido el va a atacar primero y se repite la lógica previamente explicada.

```
        if (!hero->isAlive() or !enemy->isAlive()){
            continueBattle = false;
        }
        cout << "End of the turn " << nOfTurns << endl << endl;
```

Al final de cada turno se revisa si alguien ha muerto, en caso que si, se sale de la batalla.

```
if (nOfTurns>=30){
            cout << "The hero got tired and was defeated by the " <<
enemy->getName()
            << endl << endl;
            continueBattle=false;
            winner = enemy;
        }
```

Pequeña comprobación para cuando la pelea aparenta continuar por la eternidad para que se evite ese loop y se salga de la pelea.
Estos casos es cuando los personajes hagan muy poco daño que se cancelen con el *shield*, y se atacan infinitamente, por lo tanto es necesario este añadido.

```
    if (hero->isAlive() &! enemy->isAlive()){
        won = true;
        winner = hero;
    }
    else{
        won = false;
        winner = enemy;
    }
```

Comprobación de quien fue el ganador.

Fin de los cambios de mi código.