Trabalho Prático
Fase 1 - Classes

LESI

Programação
Orientada a Objetos

Docente: Ernesto Carlos Casanova Ferreira

Autor: Nuno Rodrigo Rebelo Sá Silva, 28005

Data: 15/11/2024

Repositório: Github

Índice

Introdução

O objetivo deste projeto é implementar um sistema de classes para um jogo de RPG, seguindo princípios da programação orientação a objetos como:

- Herança;
- Encapsulamento;
- Abstração.

A hierarquia das classes foi projetada para representar diferentes tipos de personagens, cada um com habilidades únicas e mecânicas personalizadas. O foco principal está na implementação de superclasses e subclasses, utilizando abstração e para garantir flexibilidade e reutilização de código.

Superclasse (RPG/Classes/BaseStats.cs)

Public abstract class Class{

A classe Class é uma classe abstrata que define os principais atributos e comportamentos comuns a todos os personagens. Os métodos abstratos garantem que cada classe implementa funcionalidades específicas.

- **Atributos Principais**:
  - HP, Defense, Strength: Estatísticas básicas de cada personagem.
  - Stamina, UltimateGaugeSize: Gerenciam recursos e habilidades especiais.
  - Name, Type: Caracterizam o nome e o tipo de classe.
- **Métodos Abstratos**:
  - Attack(), Special(), Ultimate(), Block(): Definem as ações disponíveis para os personagens.
  - GetGauge(): Gerencia o recurso principal de cada classe (e.g., Mana, Rage).
- **Método Protegido**:
  - RechargeGauge(int value): Usado para recarregar o recurso principal.

- Representação das propriedades

```
#region Properties
private string Name { get; set; }
private string Type { get; set; }
private int HP { get; set; }
private int OriginalDefense { get; set; }
private int Defense { get; set; }
private int Strength { get; set; }
private int Stamina { get; set; }
private int UltimateGaugeSize { get; set; }
#endregion
```

- Representação do Construtor

```
#region Constructor
public Class(string name, string type, int hp, int defense, int strength, int stamina, int ultimateGaugeSize)
{
   this.Name = name;
   this.Type = type;
   this.HP = hp;
   this.Defense = defense;
   this.OriginalDefense = defense;
   this.Strength = strength;
   this.Stamina = stamina;
   this.UltimateGaugeSize = ultimateGaugeSize;
}
#endregion
```

- Representação dos injetores (devido a propriedades privadas)

```
#region Injectors
 public string name
 {
    get { return this.Name; }
    set { this.Name = value; }
 }
 public string type
 {
    get { return this.Type; }
    set { this.Type = value; }
 }
 public int hp
 {
    get { return this.HP; }
    set { this.HP = value; }
 }
 public int defense
 {
    get { return this.Defense; }
    set { this.Defense = value; }
 }
 public int originalDefense
 {
    get { return this.OriginalDefense; }
 }

public int strength
 {
    get { return this.Strength; }
    set { this.Strength = value; }
 }
 public int stamina
 {
    get { return this.Stamina; }
    set { this.Stamina = value; }
 }
 #endregion
```

Métodos

- Não abstratos:
    1. Verificação se o personagem está vivo

```
public bool alive
{
        get { return this.HP > 0; }
}
```

    2. Recarregar recurso principal de forma protegida

```
protected int RechargeGauge(int value)
{
        int currentGauge = GetGauge();
        currentGauge += value;
        if (currentGauge > this.UltimateGaugeSize)
        {
        currentGauge = this.UltimateGaugeSize;
        }
        return currentGauge;
}
```

- Abstratos:

```
public abstract int Attack();
public abstract int Special();
public abstract int Ultimate();
public abstract void Block();
```

Subclasses (RPG/Classes/Classes.cs)

As subclasses representam diferentes tipos de personagens, implementando as especificidades das habilidades:

Warrior

- Recurso: Rage.
- Estilo: Alta força e defesa; habilidades consomem e recarregam Rage.

Paladin

- Recurso: Holy.
- Estilo: Balanceado, com foco em defesa e habilidades de luz.

Assassin

- Recurso: Stealth e Dexterity.
- Estilo: Ataques rápidos e baseados em furtividade.

Archer

- Recurso: Arrows e Dexterity.
- Estilo: Ataques à distância com dano baseado em destreza.

Swordsman

- Recurso: Focus.
- Estilo: Força bruta com ataques consistentes.

Mage

- Recurso: Mana.
- Estilo: Dano mágico baseado em Magic.
- Destaque:
    - Attack(): Baseado em magia, não em força física.

Representação das subclasses

Todas as subclasses seguem a seguinte estrutura:

- Subclass
  - Propriedade
  - Construtor
  - Funções/Métodos abstratos

Representação em código:

```
public class Warrior : Class
{
    private int Rage { get; set; }

    public Warrior(int hp, int defense, int strength, int rage)
        : base("The Warrior", "Shield", hp, defense, strength, 100, 50)
    {
        this.Rage = rage;
    }

    public override int Attack()
    {
        this.Rage = RechargeGauge(10);
        return (int)(this.strength * 1.2
    }

    public override int Special()
    {
        int damage = (int)(this.strength * 1.5);
        this.Rage -= 20; // Consumes some rage
        return damage;
    }

    public override int Ultimate()
    {
        int damage = (int)(this.strength * 2.5);
        this.Rage = 0; // Consumes all rage
        return damage;
    }

    public override void Block()
    {
        this.Rage = RechargeGauge(8); // Recharge rage when blocking
        this.defense += 5; // Buffs defense temporarily for blocking
    }

    public override int GetGauge()
    {
        return this.Rage; // Return the current rage gauge value
    }
}
```

Classes

Equipa de Batalha (RPG/Classes/BattleTeam.cs)

A classe BattleTeams gerencia uma equipe de personagens do tipo Class. Oferece métodos para adicionar e remover personagens da equipe.

- **Atributos**:
  - Team: Lista de personagens.
  - FullTeam: Indica se a equipe está completa.
- **Métodos**:
  - Add(Class charc): Adiciona um personagem à equipe.
  - Remove(Class charc): Remove um personagem da equipe.

A representação em Código desta classe consite de um boleano que determina se a equipa se encontra cheia e a lista de personagens que fazem a equipa.

```
public class BattleTeams
{
   private bool FullTeam { get; set; }
   private List<Class> Team { get; set; }

   public BattleTeams()
   {
      this.FullTeam = true;
      this.Team = new List<Class>();
   }
   public List<Class> team { get { return this.Team; } }
   public List<Class> add(Class charc)
   {
      team.Add(charc);
      return team;
   }
   public List<Class> remove(Class charc)
   {
      team.Remove(charc);
      return team;
   }
}
```

Utilizadores (RPG/User/User.cs)

A classe User é responsável por representar os jogadores do sistema RPG. Ela encapsula as informações principais de cada usuário, como nome, estado de autenticação, vitórias e partidas realizadas.

Os seguintes atributos são utilizados para armazenar as informações de cada usuário:

- Username: Nome de usuário.
- LogedIn: Estado atual de autenticação (verdadeiro/falso).
- Wins: Número de vitórias acumuladas.
- Matches: Número total de partidas jogadas.

Representação das propriendas do utilizadores em código:

```
public class User
{
   private string Username { get; set; }
   private bool LogedIn { get; set; }
   private int Wins { get; set; }
   private int Matches { get; set; }

 }
```

Estutura da Solução