

北京郵電大學

程序说明文档



课程名称: 面向对象程序设计实践 (C++)

项目名称: 宠物小精灵对战系统

学 院: 计算机学院

班 级: 2022211312

学 号: 2022211404

姓 名: 唐梓楠

2024 年 6 月 14 日

目录

1 软件开发环境	2
2 快速上手	2
3 模块清单	2
4 图形界面设计	3
5 宠物 Pokemon 类设计	9
5.1 类设计	9
5.1.1 基类: Pokemon	9
5.1.2 派生类: AttackPokemon	12
5.1.3 派生类: Charmander	13
5.1.4 派生类: Mankey	14
5.2 测试程序	15
5.3 总结	17
6 数据库设计	17
6.1 用户表 user	17
6.2 宝可梦表 pokemon	17
7 界面设计	18
8 Socket 通信设计	19

1 软件开发环境

- 编程语言: C++ 20
- UI 开发框架: Qt 6.7.0
- 数据库: MySQL 8.3.0
- IDE: Qt Creator 13.0.2
- 系统: macOS Sonoma 14.5(23F79)

2 快速上手

1. 克隆仓库至本地, cd 进入

```
1 git clone https://github.com/Word2VecT/Pokemon.git  
2 cd Pokemon
```

2. 安装对应版本的 Qt, 并在 Qt Creator 中配置

3. 安装 MySQL 数据库, 并启动后台监听服务, 程序使用数据库用户名为 root, 密码为空, 数据库名为 Pokemon, 也可以自行修改 Server/main.cpp 文件配置为本地

4. 使用命令导入数据库

```
1 mysql -u root -p < ./Database/Pokemon.sql
```

5. 在 Qt Creator 中打开 Server 项目, 编译运行

6. 在 Qt Creator 中打开 Client 项目, 编译运行

7. Enjoy!

3 模块清单

表 1: 模块清单

模块名称	模块标识符	模块说明
开始窗口	MainWindow	开始界面
登陆窗口	LoginDialog	用于完成登陆、注册功能的窗口
游戏大厅	Home	游戏大厅
背包界面	Bag	显示用户所有的 Pokemon 及精灵选择
Pokemon 展示子项	ListItem	在背包或选择中展示 Pokemon 基础信息
Pokemon 信息界面	pokemonInfo	显示 Pokemon 的信息
用户列表	Rank	显示所有用户的信息
对战选择	Battle	显示对战种类选择
对战界面	Stadium	显示对战过程
结果界面	Result	显示对战结果

4 图形界面设计



图 1: 开始界面



图 2: 登录界面



图 3: 游戏大厅

	用户名	用户 ID	宠物个数	高级宠物个数	是否在线
1	123	19	1	0	在线
2	testt	3	30	0	不在线
3	q	5	16	0	不在线
4	55	11	5	0	不在线
5	zf	16	3	0	不在线
6	44	12	2	0	不在线
7	33	13	2	0	不在线
8	78	14	2	0	不在线

图 4: 用户列表



图 5: 用户信息



图 6: Pokemon 信息界面



图 7: 对战选择界面



图 8: 对战界面



图 9: 对战胜利界面

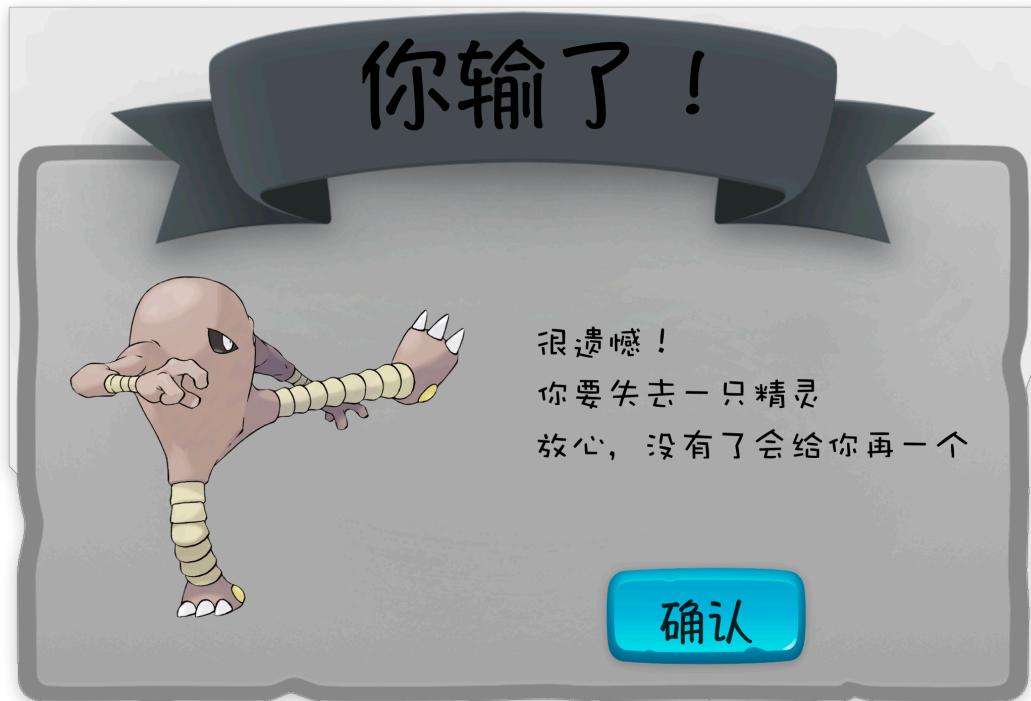


图 10: 对战失败界面

5 宠物 Pokemon 类设计

设计了宠物 Pokemon 的类, 模拟了精灵的各种属性和行为. 精灵的属性包括名字、等级、经验值、攻击力、防御力、生命值、速度、种类和能力. 每个精灵有独特的攻击方式, 等级提升时属性会相应增加. 为了方便扩展, 精灵的攻击方法设计为虚方法.

5.1 类设计

5.1.1 基类: Pokemon

Pokemon 类是所有精灵的基类, 包含以下主要属性和方法:

- 属性:
 - id: 精灵的唯一标识符
 - name: 精灵的名字
 - lv: 精灵的等级

- exp: 精灵的经验值
- atk: 精灵的攻击力
- def: 精灵的防御力
- hp: 精灵的生命值
- speed: 精灵的速度
- type: 精灵的种类
- ability: 精灵的能力

- 方法:

- 构造函数: 初始化精灵的属性
- 各种 getter 和 setter 方法: 获取和设置精灵的各个属性
- 虚函数 lvUp: 精灵升级方法 (派生类中实现)
- 虚函数 attack: 精灵攻击方法 (派生类中实现)

```
1 #ifndef POKEMON_H
2 #define POKEMON_H
3
4 #include <QString>
5 #include "config.h"
6 #include <QRandomGenerator>
7
8 // 基类: Pokemon
9 // 这个类表示一个基本的精灵, 包含了精灵的基本属性和方法
10 class Pokemon {
11     protected:
12         int id; // 精灵的唯一标识符
13
14         QString name; // 精灵的名字
15         bool evolved; // 精灵是否已进化
```

```
16  
17     int lv; // 精灵的等级  
18     int exp; // 精灵的经验值  
19  
20     int atk; // 精灵的攻击力  
21     int def; // 精灵的防御力  
22     int hp; // 精灵的生命值  
23     int speed; // 精灵的速度  
24  
25     QString type; // 精灵的种类  
26     QString ability; // 精灵的能力  
27  
28 public:  
29     // 构造函数，初始化精灵的属性  
30     Pokemon(QString name = "", int lv = 1, int atk = INIT_ATK, int def  
→      = INIT_DEF, int hp = INIT_HP, int speed = INIT_SPEED, QString  
→      ability = "");  
31  
32     // 虚析构函数  
33     virtual ~Pokemon() {};  
34  
35     // 获取精灵的名字  
36     QString getName() const;  
37     // 获取精灵的唯一标识符  
38     int getID() const;  
39     // 获取精灵的等级  
40     int getLv() const;  
41     // 获取精灵的经验值  
42     int getExp() const;  
43     // 获取精灵的攻击力
```

```
44     int getAtk() const;  
45     // 获取精灵的防御力  
46     int getDef() const;  
47     // 获取精灵的生命值  
48     int getHp() const;  
49     // 获取精灵的速度  
50     int getSpeed() const;  
51     // 获取精灵的能力  
52     QString getAbility() const;  
53  
54     // 设置精灵的等级  
55     void setLv(int);  
56     // 设置精灵的经验值  
57     void setExp(int);  
58     // 设置精灵的攻击力  
59     void setAtk(int);  
60     // 设置精灵的防御力  
61     void setDef(int);  
62     // 设置精灵的生命值  
63     void setHp(int);  
64     // 设置精灵的速度  
65     void setSpeed(int);  
66  
67     // 判断精灵是否死亡  
68     bool isDead() const;  
69  
70     // 虚函数：精灵升级，需要在派生类中实现  
71     virtual void lvUp() = 0;  
72     // 虚函数：精灵攻击另一个精灵，需要在派生类中实现  
73     virtual int attack(Pokemon*) = 0;
```

```
74     };
75
76 #endif // POKEMON_H
```

下面以力量型 Pokemon 为例, 说明 Pokemon 属性和方法的设计.

5.1.2 派生类: AttackPokemon

AttackPokemon 类继承自 Pokemon 类, 是可以攻击的精灵类. 在 AttackPokemon 类中, 重写了 attack 方法, 实现了精灵的攻击行为. 在攻击时, 精灵会根据自己的攻击力和对方的防御力计算伤害值, 然后对对方造成伤害.

lvUp: 精灵等级提升时, 属性增加, 如果精灵达到 11 级, 名字改变表示进化.

```
1 #ifndef ATTACKPOKEMON_H
2 #define ATTACKPOKEMON_H
3
4 #include "pokemon.h"
5
6 // 派生类: AttackPokemon
7 // 这个类继承自 Pokemon, 表示具有攻击能力的精灵
8 class AttackPokemon : public Pokemon {
9
10    public:
11        // 继承基类 Pokemon 的构造函数
12        using Pokemon::Pokemon;
13
14        // 虚析构函数
15        virtual ~AttackPokemon() {};
16
17        // 重写基类的纯虚函数 lvUp, 实现精灵升级的逻辑
18        void lvUp() override;
19
20        // 纯虚函数: 精灵攻击另一个精灵, 需要在派生类中实现
```

```

20         virtual int attack(Pokemon*) override = 0;
21     };

```

5.1.3 派生类: Charmander

Charmander 类继承自 AttackPokemon, 表示小火龙精灵. 重写了 attack 方法, 定义小火龙的攻击逻辑:

attack: 计算攻击伤害并随机生成攻击效果, 如果小火龙等级大于 10 且触发灼烧效果, 额外增加伤害.

```

1 Charmander::Charmander(QString name, int lv, int atk, int def, int hp, int
2   → speed) : AttackPokemon(name, lv, atk, def, hp, speed, "火花") {}
3
4     int Charmander::attack(Pokemon* enemy) {
5
6       int damage = atk - enemy->getDef();
7
8       QRandomGenerator *generator = QRandomGenerator::global();
9
10      int random = generator->bounded(6);
11
12      if (random <= 2) {
13
14        damage *= random;
15
16      }
17
18
19      if (getLv() > 10 && generator->bounded(100) < getSpeed()) {
20
21        damage += 0.5 * getDef();
22
23      } //灼烧效果
24
25      enemy->setHp(fmax(enemy->getHp() - damage, 0));
26
27
28      return random;
29
30    }

```

5.1.4 派生类: Mankey

Mankey 类继承自 AttackPokemon, 表示猴怪精灵. 重写了 attack 方法, 定义猴怪的攻击逻辑:

attack: 计算攻击伤害并随机生成攻击效果, 如果猴怪等级大于 10 且触发防御降低效果, 降低对方防御并增加自身攻击力.

```
1 Mankey::Mankey(QString name, int lv, int atk, int def, int hp, int speed) :
2     ↵     AttackPokemon(name, lv, atk, def, hp, speed, "踢倒") {}
3
4     int Mankey::attack(Pokemon* enemy) {
5         int damage = atk - enemy->getDef();
6         QRandomGenerator *generator = QRandomGenerator::global();
7         int random = generator->bounded(6);
8         if (random <= 2) {
9             damage *= random;
10
11         if (getLv() > 10 && generator->bounded(100) < getSpeed()) {
12             enemy->setDef(enemy->getDef() - 0.5 * getAtk());
13             setAtk(getAtk() + 0.5 * enemy->getDef());
14             } //降低敌方防御
15
16             enemy->setHp(fmax(enemy->getHp() - damage, 0));
17             return random;
18 }
```

5.2 测试程序

```
1 #include <QCoreApplication>
2 #include "pokemon.h"
3 #include "attackpokemon.h"
4 #include <iostream>
5
6 void printPokemonStats(const Pokemon& pokemon) {
```

```
7     std::cout << "Name: " << pokemon.getName().toStdString() <<
8         std::endl;
9     std::cout << "Level: " << pokemon.getLv() << std::endl;
10    std::cout << "Attack: " << pokemon.getAtk() << std::endl;
11    std::cout << "Defense: " << pokemon.getDef() << std::endl;
12    std::cout << "HP: " << pokemon.getHp() << std::endl;
13    std::cout << "Speed: " << pokemon.getSpeed() << std::endl;
14    std::cout << "Ability: " << pokemon.getAbility().toStdString() <<
15        std::endl;
16
17    int main(int argc, char *argv[])
18    {
19        QCoreApplication a(argc, argv);
20
21        // Create some Pokemon instances
22        Charmander charmander("Charmander", 5, 52, 43, 39, 65);
23        Mankey mankey("Mankey", 5, 80, 35, 40, 70);
24
25        // Print initial stats
26        std::cout << "Initial Stats of Charmander:" << std::endl;
27        printPokemonStats(charmander);
28
29        std::cout << "\nInitial Stats of Mankey:" << std::endl;
30        printPokemonStats(mankey);
31
32        // Attack each other
33        std::cout << "\nCharmander attacks Mankey!" << std::endl;
```

```
34     charmander.attack(&mankey);
35     printPokemonStats(mankey);
36
37     std::cout << "\nMankey attacks Charmander!" << std::endl;
38     mankey.attack(&charmander);
39     printPokemonStats(charmander);
40
41     // Level up Charmander
42     std::cout << "\nCharmander levels up!" << std::endl;
43     charmander.lvUp();
44     printPokemonStats(charmander);
45
46     // Level up Mankey
47     std::cout << "\nMankey levels up!" << std::endl;
48     mankey.lvUp();
49     printPokemonStats(mankey);
50
51     return 0;
52
53     return a.exec();
54 }
```

5.3 总结

通过上述设计, 精灵类的继承和多态性得到了良好的体现。每种精灵都有其独特的属性和攻击方式, 且可以在此基础上进一步扩展新的精灵种类和攻击逻辑。通过测试程序, 可以验证各个类及其方法的正确性, 确保精灵在游戏中的行为符合设计预期。

6 数据库设计

6.1 用户表 user

表 2: 用户表 user

表项	含义
userID	用户 ID
username	用户名
password	密码
online	在线状态
pokemonCnt	宝可梦数量
highPokemonCnt	高级宝可梦数量
battleCnt	战斗次数
battleWinCnt	战斗胜利次数

6.2 宝可梦表 pokemon

表 3: 宝可梦表 pokemon

表项	含义
pokemonID	宝可梦 ID
name	名称
HP	生命值
ATK	攻击力
DEF	防御力
LV	等级
Speed	速度
userID	主人用户 ID

7 界面设计

Qt 的使用也体现了面向对象的编程思想：

- **类和对象：**Qt 广泛使用类和对象来封装数据和功能。每个控件、窗口、对话框等都被封装成类，通过实例化类来创建对象，这些对象负责管理自身的状态和行为。

继承：Qt 利用继承机制来扩展和定制现有的类。通过继承，可以创建新的类并添加或修改功能，而不需要从头开始编写代码。Qt 中许多控件都是从 `QWidget` 或 `QAbstractItemModel` 等基类继承而来的。

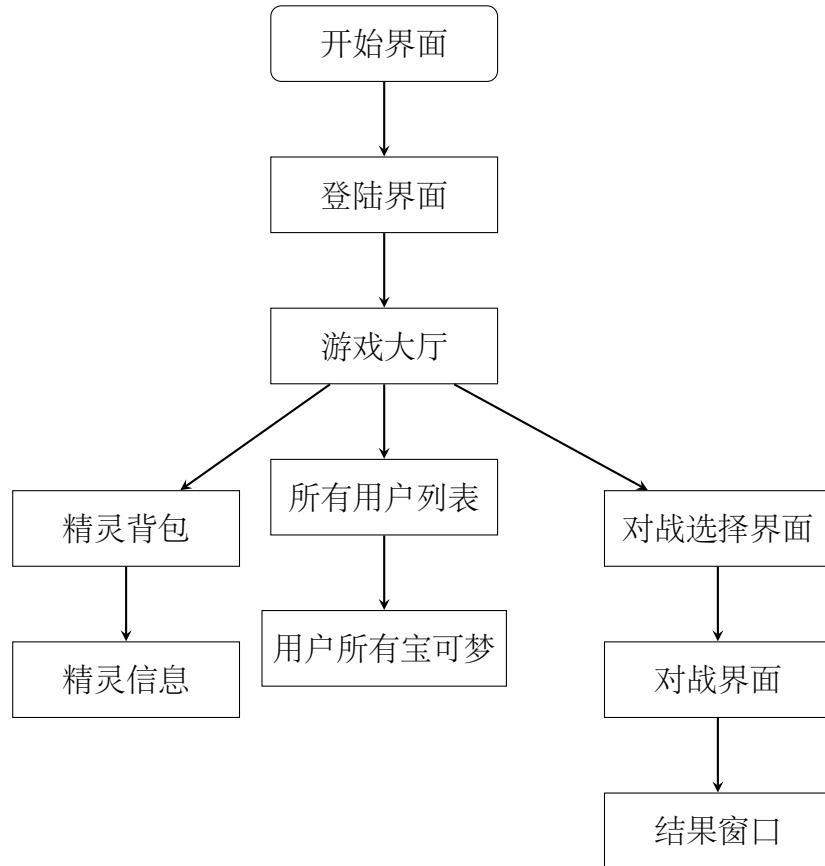
多态性：Qt 通过虚函数和接口实现多态性，使得程序可以在运行时选择合适的函数执行。这样可以实现灵活和可扩展的设计。

信号与槽机制：这是 Qt 的一大特色，用于实现对象之间的通信。对象可以通过信号通知其他对象发生了某个事件，而其他对象可以通过槽函数来处理这些事件。这种机制使得对象之间的耦合度降低，提高了代码的可维护性和可扩展性。

属性系统：Qt 提供了一个强大的属性系统，使得对象的属性可以被动态查询和修改。属性系统与信号槽机制结合，可以实现复杂的交互和动态行为。

事件处理：Qt 中所有的用户交互（如鼠标点击、键盘输入等）都被封装成事件，通过事件循环进行处理。对象可以通过重写事件处理函数来定制其行为。

通过这些面向对象的设计思想，Qt 提供了一个高效、灵活且可扩展的应用开发框架，使得我可以专注于实现应用的业务逻辑，而不用过多关心底层的实现细节。



8 Socket 通信设计

采用了多客户端并发的 C/S 模式, 这样服务器可以和多个客户端通信。采用的方法是单线程的方法。在服务器接收到新的客户端连接时, 服务器为该客户端分配一个新的 socketID, 并且将 socketID 传回给客户端。与此同时, 服务器将套接字加入到服务器的套接字列表中。

在客户端与服务器通信的过程中, 客户端在传给服务器的信息中加入 socketID, 服务器收到客户端传来的信息, 并且做相应的处理之后, 将该 socketID 放入到传给客户端的信息中, 并且将该信息发送给服务器 socket 列表中的所有客户端。客户端收到信息之后先检查 socketID 是否匹配, 若匹配则接收信息, 否则不接收。