**北京邮电大学课程设计报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程设计**  **名称** | 面向对象程序设计与实践2 | **学 院** | 计算机学院 | **指导教师** | 双锴 |
| 学生姓名 | 罗昊中 | 班 级 | 2016211306 | 学 号 | 2016211282 |
| 课  程  设  计  内  容 | 本课程设计要求利用C++面向对象思想设计一个小精灵在线对战系统。共包含三个版本，第一版本设计小精灵类，第二版本加入服务端实现网络通信与用户信息管理，第三版本实现用户在线对战。本实验由单人独立完成。 | | | | |
| **学生**  **课程设计**  **报告** | 见程序代码。 | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | 遵照实践教学大纲并根据以下四方面综合评定成绩：  1、课程设计目的任务明确，选题符合教学要求，份量及难易程度  2、团队分工是否恰当与合理  3、综合运用所学知识，提高分析问题、解决问题及实践动手能力的效果  4、是否认真、独立完成属于自己的课程设计内容，课程设计报告是否思路清晰、文字通顺、书写规范  **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | |

第一版本

1. 问题陈述

设计宠物小精灵的类，为简化游戏设计，精灵的属性包括种类（力量型：高攻击； 肉盾型：高生命值； 防御型：高防御； 敏捷型：低攻击间隔，共四种）、名字、等级、经验值、攻击力、防御力、生命值、攻击间隔等。

每个精灵初始等级为1，满级15级，每当精灵升级的时候，宠物对应的属性值会有少量增加（主属性增加量相对较多）。

设计一个精灵的基类，并为每种小精灵设计攻击方法。

请写一个测试程序对设计的精灵类的相关属性和方法（包括攻击函数，升级函数等）进行测试。

运行环境：win10

1. 名次提取

宠物小精灵 属性 种类 名字 等级 经验值 攻击力 防御力 生命值 攻击间隔 初始等级 属性值 攻击方法 测试程序 攻击函数 升级函数

筛选后，类：宠物小精灵 测试程序

属性：属性 种类 名字 等级 经验值 攻击力 防御力 生命值 攻击间隔 初始等级

方法：攻击方法 攻击函数 升级函数

1. 类设计

小精灵基类：

class pet

{

private:

QString name;

petKind kind;

petGrade grade;

int lv;

int xp;

int xpMax;

int hp;

int atk;

int def;

int atkInterval;

int missRate;

//when figiting

int hpNow;

int atkNow;

int defNow;

int atkIntervalNow;

int missRateNow;

int attackTime;

public:

pet(petKind \_kind, QString \_name, int \_lv);

virtual ~*pet*();

petKind getKind() const { return kind; };

petGrade getGrade() const { return grade; };

QString getName() const { return name; };

int getLv() const { return lv; };

int getXp() const { return xp; };

int getXpMax() const { return xpMax; };

int getHp() const { return hp; };

int getAtk() const { return atk; };

int getDef() const { return def; };

int getAtkInterval() const { return atkInterval; };

int getHpNow() const { return hpNow; };

int getAtkNow() const { return atkNow; };

int getDefNow() const { return defNow; };

int getAtkIntervalNow() const { return atkIntervalNow; };

int getMissRateNow() const { return missRateNow; };

int getAttackTime() const { return attackTime; };

void changeAtkNow(int \_atk) { atkNow = \_atk; };

void changeDefNow(int \_def) { defNow = \_def; };

void changeHpNow(int \_hp) { hpNow = \_hp; };

void changeAtkIntervalNow(int \_atkInterval) { atkIntervalNow = \_atkInterval; };

void lvUp();

void lvInitialize();

//void gradeUp();

void fightStart();

void fightEnd(int experience);

void beAttacked(QTextBrowser \*p , const pet &attacker);

void print(QPainter &p ,int x ,int y);

virtual void *performAbilities*(QTextBrowser \*p, pet &attacked) = 0;

};

小精灵派生类：

class smallFireDragon: public pet

{

public:

smallFireDragon(petKind , QString , int);

smallFireDragon(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

class pikachu: public pet

{

public:

pikachu(petKind , QString , int);

pikachu(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p, pet &attacked);

};

class bigStone: public pet

{

public:

bigStone(petKind , QString , int);

bigStone(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

class oldTree: public pet

{

public:

oldTree(petKind , QString , int);

oldTree(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

控制类：

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

public slots:

void lvUpPet();

void fighting();

protected:

void *paintEvent*(QPaintEvent\*e);

private:

Ui::MainWindow \*ui;

QPixmap \*pic1;

QPixmap \*pic2;

QPushButton \*lvUpButton;

QPushButton \*fightButton;

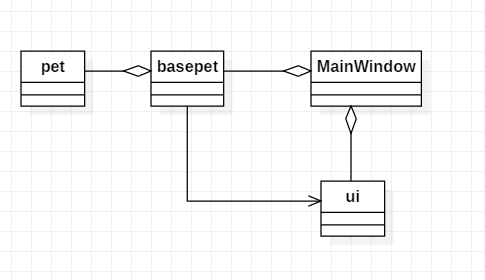
QTextBrowser \*out;

pet \*pPet;

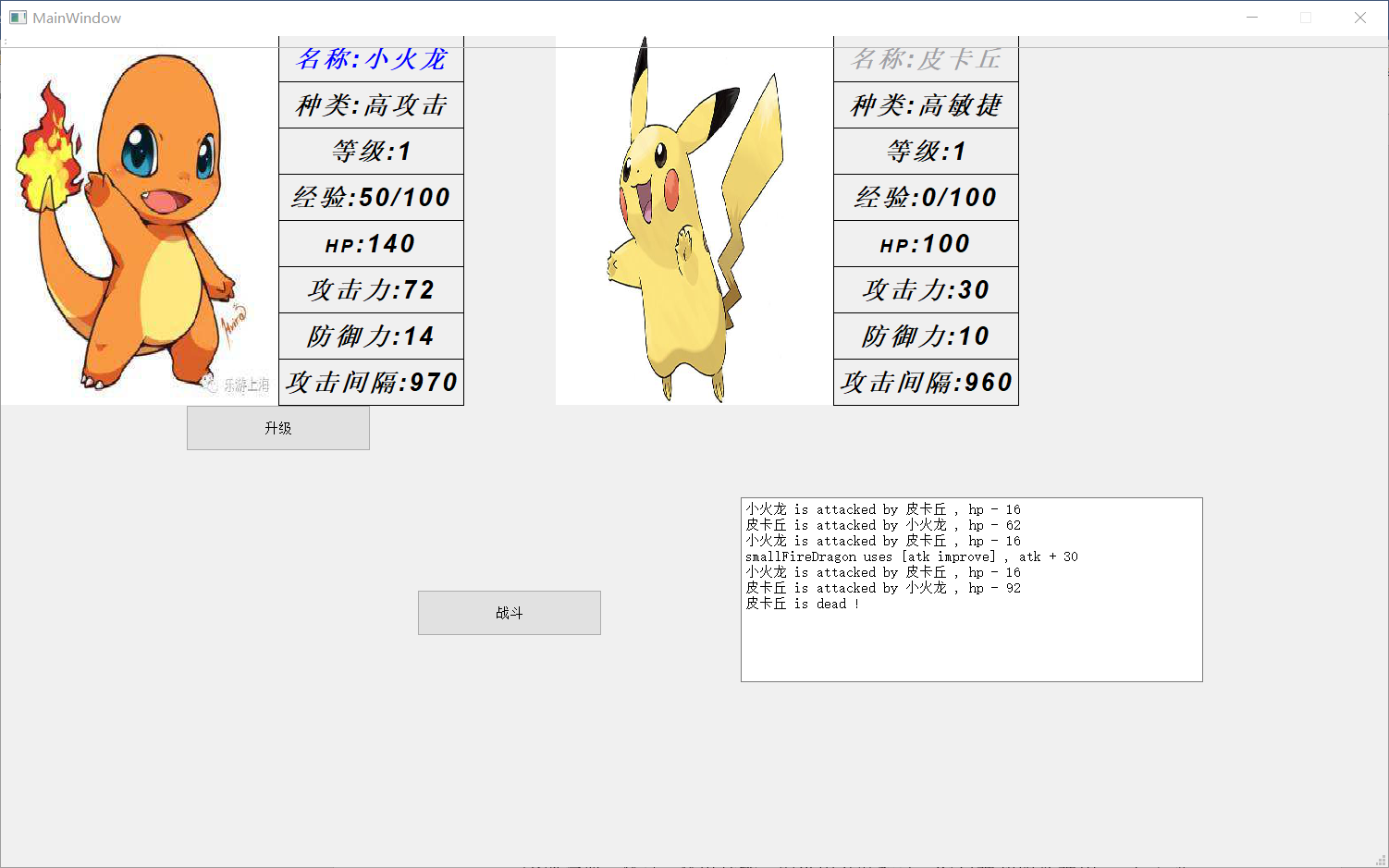
pet \*pPet1;

};

四．类图



五．运行截图



第二版本

一．问题陈述

在第一版本的基础上，添加以下功能：

每个用户需要注册一个账号，用户名全局唯一，不能有任何两个用户名相同，要考虑注册失败的场景时的反馈

实现注册、登录、登出功能，均采用C/S模式，客户端和服务端用socket进行通信，服务端保存所有用户的信息（文件存储或数据库均可，数据库有额外加分）

每个用户拥有：用户名、拥有的精灵，两个属性。 用户注册成功时，系统自动随机分发三个1级精灵给用户

用户可以查看所有成功注册用户拥有的精灵，也可以查看所有当前在线的用户。

二．类设计

增添了用户类，数据库类，网络通信类，并对第一版本的小精灵类进行了改进

小精灵基类：

class pet

{

private:

petKind kind;

petGrade grade;

int lv;

int xp;

int xpMax;

int hp;

int atk;

int def;

int atkInterval;

int missRate;

//when figiting

int hpNow;

int atkNow;

int defNow;

int atkIntervalNow;

int missRateNow;

int attackTime;

public:

pet(petKind \_kind, QString \_name, int \_lv);

pet(QByteArray &b , int &pos);

virtual ~*pet*();

petKind getKind() const { return kind; };

petGrade getGrade() const { return grade; };

QString getName() const { return name; };

int getLv() const { return lv; };

int getXp() const { return xp; };

int getXpMax() const { return xpMax; };

int getHp() const { return hp; };

int getAtk() const { return atk; };

int getDef() const { return def; };

int getAtkInterval() const { return atkInterval; };

int getHpNow() const { return hpNow; };

int getAtkNow() const { return atkNow; };

int getDefNow() const { return defNow; };

int getAtkIntervalNow() const { return atkIntervalNow; };

int getMissRateNow() const { return missRateNow; };

int getAttackTime() const { return attackTime; };

void changeAtkNow(int \_atk) { atkNow = \_atk; };

void changeDefNow(int \_def) { defNow = \_def; };

void changeHpNow(int \_hp) { hpNow = \_hp; };

void changeAtkIntervalNow(int \_atkInterval) { atkIntervalNow = \_atkInterval; };

void lvUp();

void lvInitialize();

//void gradeUp();

void fightStart();

void fightEnd(int experience);

void beAttacked(QTextBrowser \*p , const pet &attacker);

void print(QPainter &p ,int x ,int y);

QString toQString();

void getPetInfo(QByteArray &b , int &pos);

//void printInFighting(QPainter &p ,int x ,int y);

virtual void *performAbilities*(QTextBrowser \*p, pet &attacked) = 0;

protected:

QString name;

};

小精灵派生类：

class smallFireDragon: public pet

{

public:

smallFireDragon(petKind , QString , int);

smallFireDragon(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

class pikachu: public pet

{

public:

pikachu(petKind , QString , int);

pikachu(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p, pet &attacked);

};

class bigStone: public pet

{

public:

bigStone(petKind , QString , int);

bigStone(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

class oldTree: public pet

{

public:

oldTree(petKind , QString , int);

oldTree(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

用户类：

class player{

private:

QString name;

QString password;

int winNum;

int failNum;

int petNum;

int highPetNum;

public:

player();

player(QByteArray &b);

~player();

vector<pet \*> petOfPlayer;

void addPet(pet \*);

void getPlayerInfo(QByteArray &b , int &pos);

void getOtherPlayer(QByteArray &b , int &pos);

QString toQString();

QString getName() const { return name; } ;

QString getPassword() const { return password; } ;

int getWinNum() const { return winNum; } ;

int getFailNum() const { return failNum; } ;

int getPetNum() const { return petNum; } ;

int getHighPetNUm() const { return highPetNum; } ;

};

客户端控制类（聚合网络通信与界面）：

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

QString IP;

int port;

player user;

unsigned int nowPet;

pet \*p;

private slots:

void socket\_disconnected();

void on\_loginIn\_clicked();

void on\_registe\_clicked();

void readData();

void on\_pushButton\_next\_1\_clicked();

void on\_pushButton\_next\_2\_clicked();

void on\_pushButton\_loginOut\_clicked();

void on\_pushButton\_lvUpBattle\_clicked();

void on\_pushButton\_battle\_clicked();

private:

Ui::MainWindow \*ui;

QTcpSocket \*socket;

};

服务端控制类（聚合网络通信）：

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

void newConnect();

void socketReadData();

void socketDisconnected();

Database playerDatabase;

vector<player\*> inLinePlayer;

private:

Ui::MainWindow \*ui;

QTcpServer \*server;

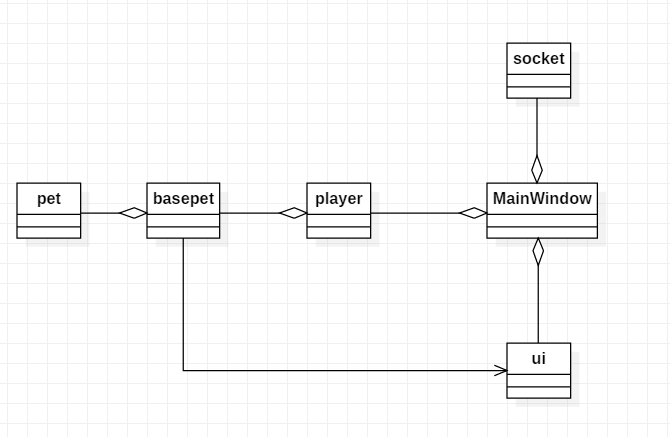
QTcpSocket \*socket;

int port;

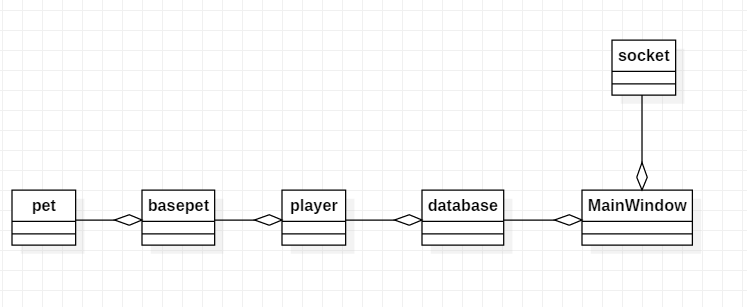
};

三．类图

客户端类图：



服务端类图：



四．通信协议设计

客户端发送：

登录时，发送消息首字母为l，注册时首字母为r，退出时首字母为o，注册登录时后面接发送用户名与密码，退出时后面接用户所有信息

服务端发送：

登录时用户名已存在，发送 "usernameisin"

注册成功时发送 "registesuccessful"

登录失败时发送 "logininfailed"

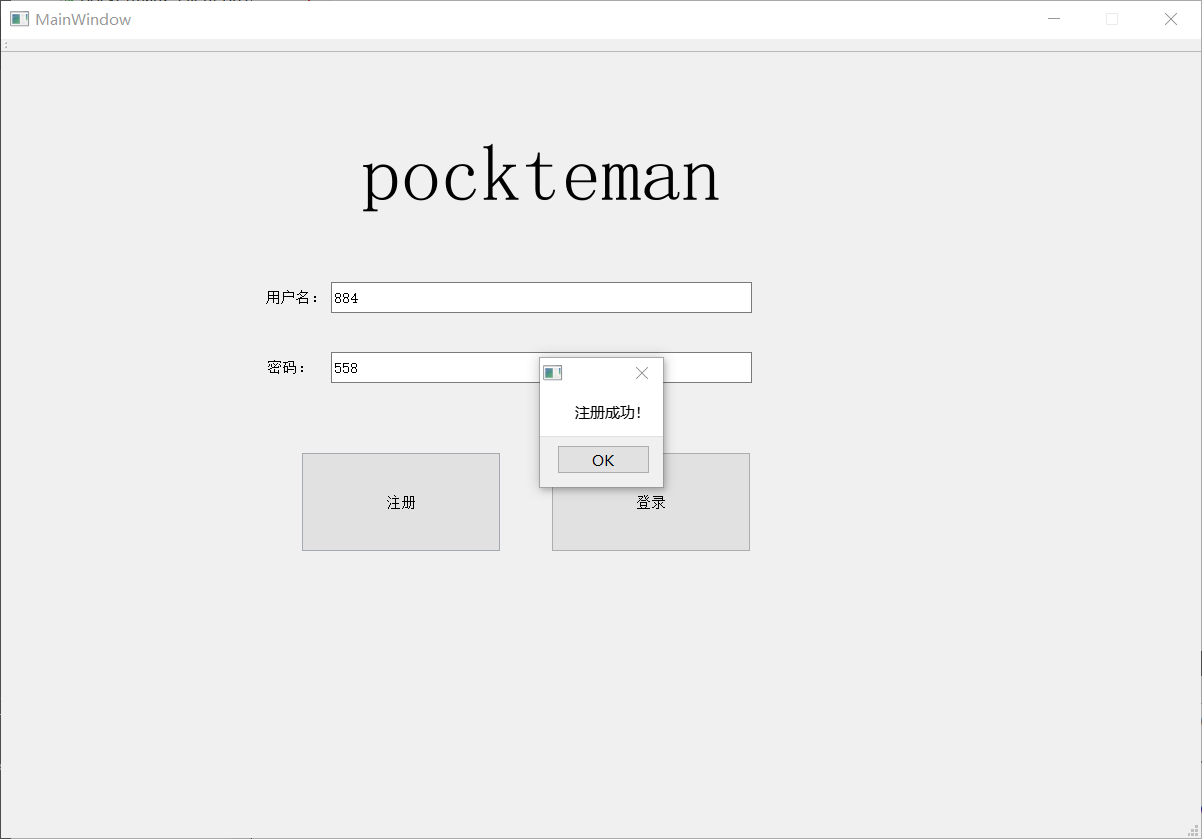
登录成功时发送 "logininsuccessful" ，后面+空格+用户信息+“ # ”+当前其他在线用户信息，每个其他在线用户信息之间用“ # ”分隔

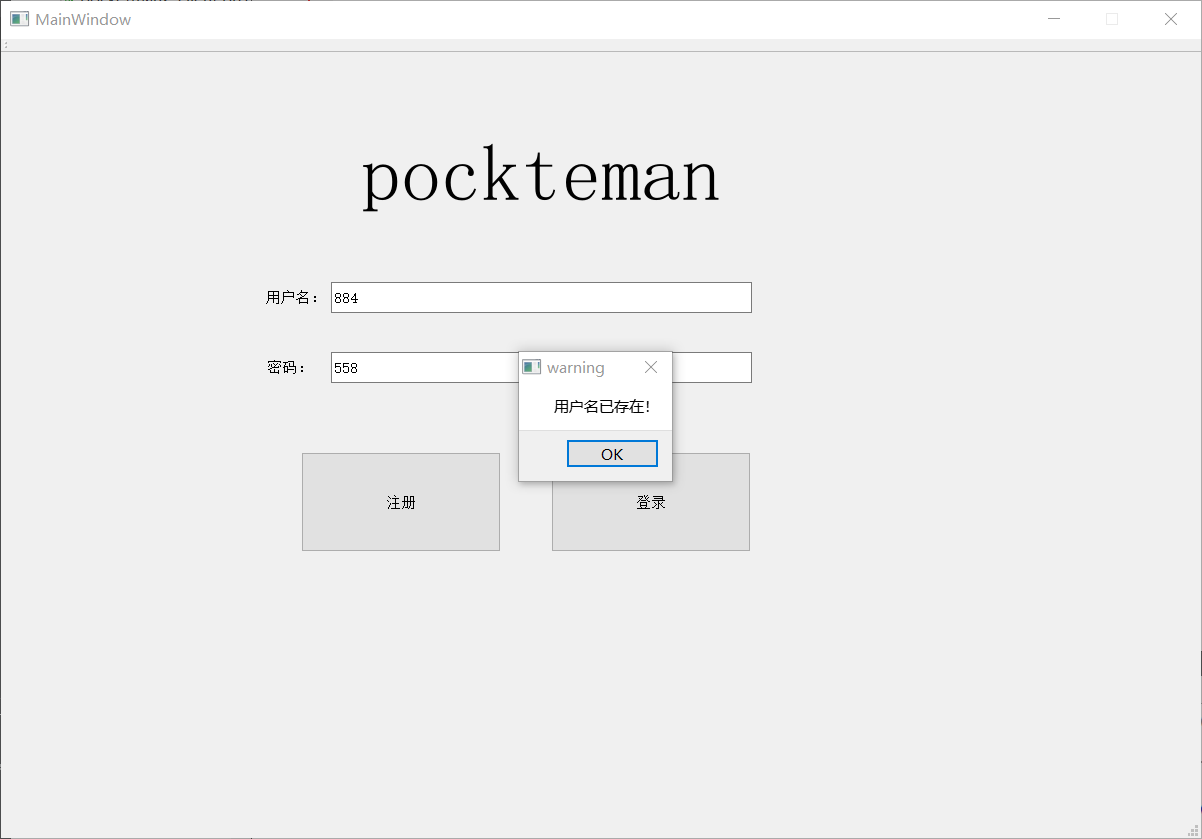
用户信息格式：用户名 密码 胜利次数 失败次数 小精灵数量 高级小精灵数量

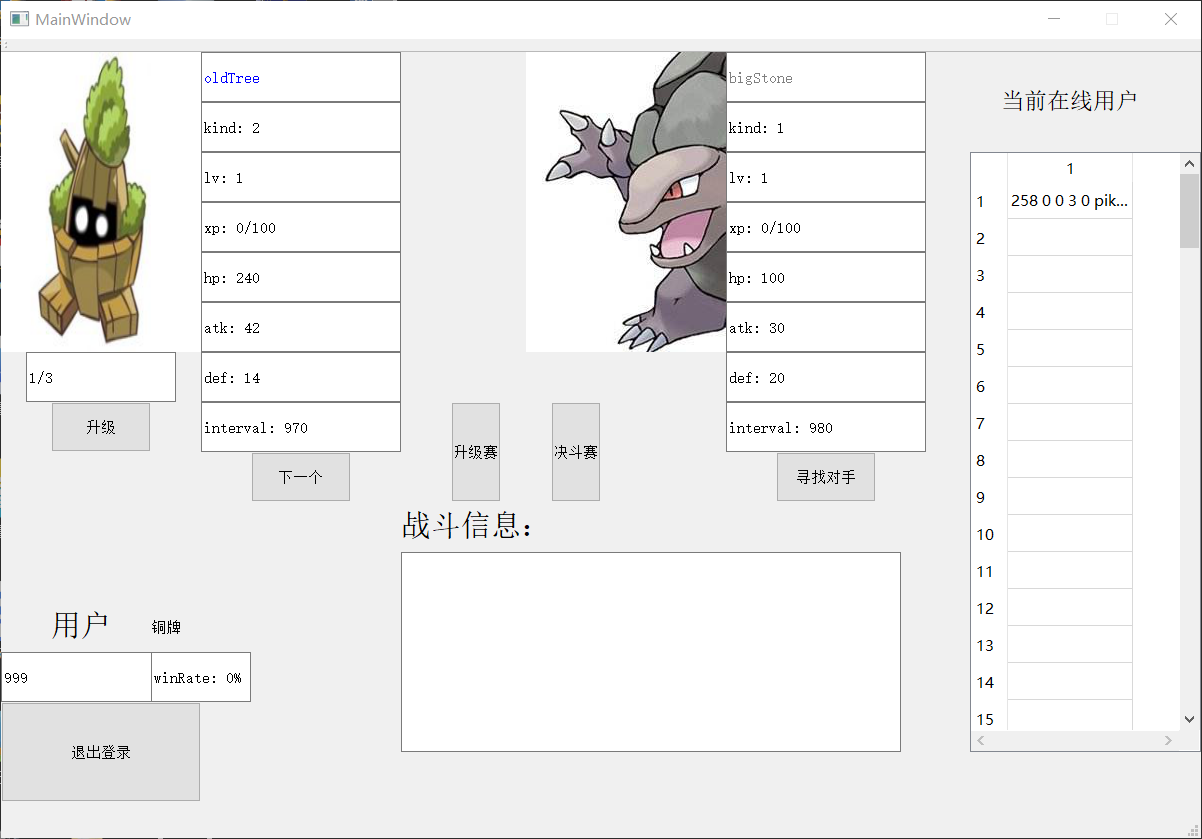
用户拥有的所有小精灵信息。每个之前用空格分隔

小精灵信息格式：名称 种类 品阶 等级 经验

五．运行截图







第三版本

1. 问题陈述

已经登录的在线用户可以和服务器进行虚拟决斗，决斗分两种：升级赛和决斗赛，两种比赛都能增长宠物经验值。服务器上有一个虚拟精灵的列表，用户可以挑选其中任意一个进行比赛（升级赛或者决斗赛）。另外决斗赛中用户胜出可以直接获得该战胜的的精灵，失败则系统从用户的精灵中随机选三个（不够三个精灵的情况就选择他所有的精灵），然后由用户选一个送出。

* 1. 升级赛 只是用户用来增加精灵经验值，规则开发者自定；
  2. 累积多少经验值升一级，规则开发者自定；
  3. 决斗赛的上述规则同升级赛，只是额外还可以赢得宠物一个。

用户如果没有精灵（比如总是失败，已经全部送出去），则系统会随机放给给他一个初级精灵。

请让你的系统自动模拟每场比赛的每次出招。另外，为了增加不确定性，可以加入概率闪避攻击和暴击伤害机制

比赛的过程和结果由系统根据上述规则自动模拟完成，要求结果具有一定的随机性。

用户增加新功能，可以查看某个用户的胜率

用户增加新属性，为宠物个数徽章（金银铜）和高级宠物徽章（金银铜），分别根据拥有的宠物个数的多少和拥有高级宠物（15级）个数的多少颁发

1. 类的设计

小精灵基类：

class pet

{

private:

petKind kind;

petGrade grade;

int lv;

int xp;

int xpMax;

int hp;

int atk;

int def;

int atkInterval;

int missRate;

//when figiting

int hpNow;

int atkNow;

int defNow;

int atkIntervalNow;

int missRateNow;

int attackTime;

public:

pet(petKind \_kind, QString, int \_lv);

pet(QByteArray &b , int &pos);

virtual ~*pet*();

petKind getKind() const { return kind; };

petGrade getGrade() const { return grade; };

QString getName() const { return name; };

int getLv() const { return lv; };

int getXp() const { return xp; };

int getXpMax() const { return xpMax; };

int getHp() const { return hp; };

int getAtk() const { return atk; };

int getDef() const { return def; };

int getAtkInterval() const { return atkInterval; };

int getHpNow() const { return hpNow; };

int getAtkNow() const { return atkNow; };

int getDefNow() const { return defNow; };

int getAtkIntervalNow() const { return atkIntervalNow; };

int getMissRateNow() const { return missRateNow; };

int getAttackTime() const { return attackTime; };

void changeAtkNow(int \_atk) { atkNow = \_atk; };

void changeDefNow(int \_def) { defNow = \_def; };

void changeHpNow(int \_hp) { hpNow = \_hp; };

void changeAtkIntervalNow(int \_atkInterval) { atkIntervalNow = \_atkInterval; };

void lvUp();

void lvInitialize();

//void gradeUp();

void fightStart();

void fightEnd(int experience);

void beAttacked(QTextBrowser \*p , const pet &attacker);

void print(QPainter &p ,int x ,int y);

QString toQString();

void getPetInfo(QByteArray &b , int &pos);

//void printInFighting(QPainter &p ,int x ,int y);

virtual void *performAbilities*(QTextBrowser \*p, pet &attacked) = 0;

protected:

QString name;

};

小精灵派生类：

class smallFireDragon: public pet

{

public:

smallFireDragon(petKind , QString , int);

smallFireDragon(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

class pikachu: public pet

{

public:

pikachu(petKind , QString , int);

pikachu(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p, pet &attacked);

};

class bigStone: public pet

{

public:

bigStone(petKind , QString , int);

bigStone(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

class oldTree: public pet

{

public:

oldTree(petKind , QString , int);

oldTree(QByteArray &b , int &pos);

virtual void *performAbilities*(QTextBrowser \*p , pet &);

};

用户类：

class player{

private:

QString name;

QString password;

int winNum;

int failNum;

int petNum;

int highPetNum;

public:

player();

player(QByteArray &b);

~player();

vector<pet \*> petOfPlayer;

void addPet(pet \*);

void getPlayerInfo(QByteArray &b , int &pos);

void getOtherPlayer(QByteArray &b , int &pos);

QString toQString();

void win() { winNum++; } ;

void fail() { failNum++; } ;

void changePetNum(int num) { petNum = num; } ;

QString getName() const { return name; } ;

QString getPassword() const { return password; } ;

int getWinNum() const { return winNum; } ;

int getFailNum() const { return failNum; } ;

int getPetNum() const { return petNum; } ;

int getHighPetNUm() const { return highPetNum; } ;

};

客户端控制类：

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

bool fighting();

void showInformation();

bool hasIllegal(QString s);

private slots:

void socket\_disconnected();

void on\_loginIn\_clicked();

void on\_registe\_clicked();

void readData();

void on\_pushButton\_next\_1\_clicked();

void on\_pushButton\_next\_2\_clicked();

void on\_pushButton\_loginOut\_clicked();

void on\_pushButton\_lvUpBattle\_clicked();

void on\_pushButton\_battle\_clicked();

void on\_pushButton\_throwName1\_clicked();

void on\_pushButton\_throwName2\_clicked();

void on\_pushButton\_throwName3\_clicked();

void on\_pushButton\_lvUp\_clicked();

private:

Ui::MainWindow \*ui;

QTcpSocket \*socket;

QString IP;

int port;

int select;

unsigned int nowPet;

player user;

pet \*p;

};

服务端数据库类：

class Database

{

public:

Database();

~Database();

QSqlDatabase database;

QSqlQuery \*query;

};

服务端控制类：

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

void newConnect();

void socketReadData();

void socketDisconnected();

Database playerDatabase;

vector<player\*> inLinePlayer;

private:

Ui::MainWindow \*ui;

QTcpServer \*server;

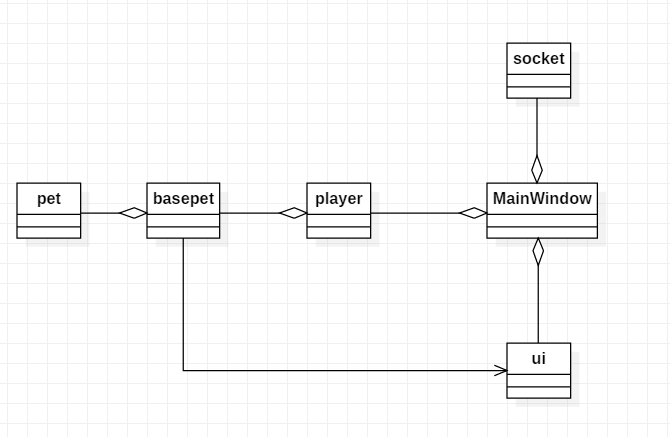
QTcpSocket \*socket;

int port;

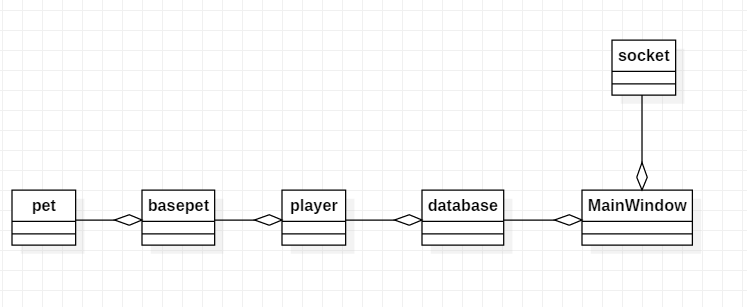
};

1. 类图

客户端：



服务端：



1. 通信协议设计

客户端发送：

登录时，发送消息首字母为l，注册时首字母为r，退出时首字母为o，注册登录时后面接发送用户名与密码，退出时后面接用户所有信息

服务端发送：

登录时用户名已存在，发送 "usernameisin"

注册成功时发送 "registesuccessful"

登录失败时发送 "logininfailed"

登录成功时发送 "logininsuccessful" ，后面+空格+用户信息+“ # ”+当前其他在线用户信息，每个其他在线用户信息之间用“ # ”分隔

用户信息格式：用户名 密码 胜利次数 失败次数 小精灵数量 高级小精灵数量 用户拥有的所有小精灵信息。每个之间用空格分隔

小精灵信息格式：名称 种类 品阶 等级 经验，之间用空格分隔

1. 数值设计

伤害计算：

攻击方小精灵攻击力减去被攻击方小精灵防御力即为所受伤害

属性计算：

atk = lv \* atkRate\_lv + grade \* lv \* atkRate\_grade;

def = lv \* defRate\_lv + grade \* lv \* defRate\_grade;

hp = lv \* hpRate\_lv + grade \* lv \* hpRate\_grade;

atkInterval = 1000 - lv \* intervalRate\_lv - grade \* lv \* intervalRate\_grade;

missRate = 10 + grade;

然后对应属性增加 lv\*rate

经验计算：

战斗胜利一场经验+50，失败经验+20

每级的升级经验都为100，满级15级

1. 运行截图

