**链表类模板应用——WarChess项目报告**

**2023年10 月1 日**

# **1 warchess项目概述**

## **1.1 项目内容及要求**

项目内容：基于链表模板衍生出的战旗类对战游戏。该游戏主要玩法为玩家通过使用一定数量的金币购买随从，花费金币为随从添加Buff，来创造属于自己的部队，在战斗中击败对手取得胜利。

程序运行基本逻辑：程序从txt文件中读取预设数据，弹出基本ui界面，界面中玩家可选择进入单人游戏，双人游戏以及设置三大主要空间，在战斗过程中程序自动处理玩家输入的数据（购买的随从及buff），通过战斗函数得出胜利者，并与战斗过程一起反馈给玩家。

## **1.2 研究人员及分工**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 学号 | 姓名 | 角色及具体贡献 |
| 1 | 22122528 | 杨弈文 | 组长（3个类和全局函数的声明与设计规划，main函数编写，Fight、Menu、Setting、ReSet、OP、WriteF、Player::ShowMinions、Minion::LoadBuff函数的编写，debug，项目报告中相关函数介绍部分，演示视频录制。） |
| 2 | 22120051 | 党昊天 | 组员（linklist头文件的解读，buy函数（链表的创建以及数据的输入），游戏内数值设计，debug（prepare函数），项目报告介绍部分，演示视频的录制） |
| 3 | 22121435 | 文婷 | 组员（玩家阵容创建Create()和RobotCreate函数,main()函数部分设计，项目报告撰写以及整合，项目测试，debug,UI界面的处理） |
| 4 | 2212 | 李宗恒 | 组员（shuru、shuchu、shuchu1函数的编写，sz.txt的格式设计和编辑） |

# **2 Buff、Minion、Player类的设计**

## **2.1 数据成员设计**

仔细阅读并研究链表模板的设计之后，本组决定采用链表模板创建双重链表，实现程序设计。

程序共由Player，Minion，Buff三个类组成，类中间通过链表模板提供对象进行串联，具体实现为，Player中存放Minion链表的头指针LinkList<Minion>，该头指针串联若干个Node<Minion>节点，所有数据（包括第二层链表的头指针Linklist<Buff>）存放于Node的data成员中（该成员类型为Minion），进而实现整个程序的运行。

### 2.1.1Buff

Buff类用于储存并添加Buff，Buff类同Minion类一样，通过静态成员保存默认Buff类型，游戏中内置了4种类型的Buff用于实际使用。此外，Buff类型可储存当前添加的Buff（通过kind对象），以及后续可选扩展功能Buff的持续时间（对象DUR\_P）。Buff类的对象通过第二重链表串联在每一个随从对象之下。

|  |
| --- |
| Public：  static int HP[11],ATK[11],DUR[11],VLU[11],CHP[11],CATK[11],CDUR[11],CVLU[11];//所有Buff类型的血量、攻击加成、持续回合数和费用  private：  int kind,DUR\_P; |

### 2.1.2Minion

随从类用于创建随从，该类有11个静态对象，表示游戏默认内置的十一种随从类型。在实际使用过程中，每个随从对象可以储存一个类型的随从，表示在战斗中使用类改类型的随从。具体实现为每个随从对象保存了所创建随从类型的编号，攻击值以及生命值（kind,HP\_P,ATK\_P;），为后续战斗过程提供基础。

在程序运行层面，随从类下还有成员buff的头指针（LinkList<Buff>h），用于表示该随从身上所添加的Buff。

|  |
| --- |
| public：  static int HP[11],ATK[11],VLU[11],CHP[11],CATK[11],CVLU[11];//所有随从的初始血量、攻击和费用  private:  int kind,HP\_P,ATK\_P;//该随从当前的血量和攻击  LinkList<Buff>h; |

### 2.2.3Player

Player类用于创建玩家，玩家拥有血量和金币对象（HP\_P,GP\_P），血量用于统计每次对战后玩家的生命值，若玩家生命值小于等于0，则该玩家便输掉了这局游戏。金币用于购买随从及Buff，游戏开始时以及每次对战结束后，都会给玩家一定量金币用于购买随从及Buff。

在程序运行层面，玩家类还有链接下层对象的头指针成员（LinkList<Minion>h），以及用于单人模式（自动生成电脑玩家）以及统计时间的成员。

|  |
| --- |
| Public:  static int GP,HP,BuyRET,FightRET1,FightRET2;  static bool ATOBuyCLS,ATOFightCLS;  private:  int HP\_P,GP\_P,REC[1001];//玩家当前剩余血量和金币数量  LinkList<Minion>h; |

## **2.2 成员函数（友元函数）原型设计**

### 2.2.1Buff类中成员函数

**构造函数、读写函数：**

|  |
| --- |
| Buff::Buff(int kind, int DUR\_P):kind(kind),DUR\_P(DUR\_P){}  int Buff::GetKind() const{return kind;} //获取属性  void Buff::SetDUR\_P(const int &x){DUR\_P=x;}  void Buff::SetKind(const int& x){kind=x;} //设置属性 |

### 2.2.2Minion类中成员函数

**构造函数、读写函数：**

|  |
| --- |
| Minion::Minion(int kind, int HP\_P, int ATK\_P):kind(kind),HP\_P(HP\_P),ATK\_P(ATK\_P){}  int Minion::GetKind() const{return kind;}  int Minion::GetHP\_P() const{return HP\_P;}  int Minion::GetATK\_P() const{return ATK\_P;} //获得相应属性  LinkList<Buff>& Minion::GetHead(){return h;}  void Minion::SetKind(const int &x){kind=x;}  void Minion::SetHP\_P(const int &x){HP\_P=x;}  void Minion::SetATK\_P(const int &x){ATK\_P=x;} //设置相应属性值 |

### 2.2.3Player类中成员函数

|  |
| --- |
| Player(int hP\_P=20,int gP\_P=10); //构造函数，默认每个玩家初始有5点血与10点金币，可修改代码或者运行程序时通过开发者模式（调用修改函数）重置数值。  int GetHP\_P() const;  int GetGP\_P() const;  int GetNumREC() const;  LinkList<Minion>& GetHead(); //获取相应属性值  void SetHP\_P(const int &x);  void SetGP\_P(const int &x); //设置相应属性值  void SetREC(const int x1,const int x2); //用于修改有效操作流日志，目前仅供调试使用，暂未被调用过。  void ShowMinions(); //用于展示所有随从以及他们身上的Buff。 |

**以下函数的代码片段中省去了部分输出语句，仅展示了逻辑语句。**

**Buy:buy\_Minion;buy\_Buff**

Buy函数集成到Create（）函数中，用于实现链表的调用，两个buy分别用于购买随从和Buff（即创建二重链表），返回值bool用于判断购买是否成功。Create函数包含引导以及对用户输入数据的判定，从而保证购买的成功进行。

|  |
| --- |
| bool Player::Buy\_Minion(const int x1){  if(Minion::VLU[x1]>GP\_P)return false;  GP\_P-=Minion::VLU[x1];  Minion \*t=new Minion;  t->SetATK\_P(Minion::ATK[x1]);  t->SetHP\_P(Minion::HP[x1]);  t->SetKind(x1);  h.Append(\*t);  return true;  }  bool Player::Buy\_Buff(const int x1,const int x2){  if(Buff::VLU[x2]>GP\_P||x1>h.NumNodes())return false;  GP\_P-=Buff::VLU[x2];  Buff \*t=new Buff;  t->SetDUR\_P(Buff::DUR[x2]);  t->SetKind(x2);  h.Go(x1-1);  h.CurData().GetHead().Append(\*t);  return true;  } |

**Create**：

本函数是Player类中成员函数，其作用是让用户创建自己的阵容。

函数具体逻辑为:最外层的while循环判断用户当前金币数量，当其等于0时，则代表用户的阵容创建完毕，反之，当其大于0时，用户可以通过我们给出的商店展示的各种随从和buff特性选择自己想要的对象，之后按照输入的格式为自己构建阵容，用户端有两个输入，若要购买随从，则第二个输入必须为0，第一个输入为想添加的随从编号；若要给随从附加buff，则两个输如分别代表已经存在的随从的序号以及想添加的buff的编号；在购买过程中，如果用户想购买的对象的价值超过自己拥有的金币数量，或者选择了一个不存在的随从添加Buff，相应的buy函数会返回布尔值false，继而通过if语句，让用户重新选择添加对象，通过以上处理，避免了出现玩家金币值为负或者Buff头指针不存在的情况出现。

|  |
| --- |
| void Player::Create()  {  int x1,x2;  while (GetGP\_P()>0)  {  cin>>x1>>x2;  if(x2==0)  {  if(Buy\_Minion(x1)==false)  continue;  else  {  if(Buy\_Buff(x1,x2)==false)  continue;  }  REC[++REC[0]]=x1;  REC[++REC[0]]=x2;  }  } |

RobotCreate：

本函数为Player类中成员函数，其作用为，让电脑自动创建人机阵容。与Create函数内部逻辑相同，唯一的区别就是将用户输入端取消，代之的为系统生成随机数。

|  |
| --- |
| void Player::RobotCreate(){  int x1,x2;  while (GetGP\_P()>0)  {  x2=rand()%5;  if(x2==0){  x1=rand()%10+1;  if(Buy\_Minion(x1)==false)  continue;  }  else{  if(h.NumNodes()==0)  continue;  x1=rand()%h.NumNodes()+1;  if(Buy\_Buff(x1,x2)==false)  continue;  }  REC[++REC[0]]=x1;  REC[++REC[0]]=x2;  }  } |

**Fight:**

友元的全局函数，用于计算战斗过程并返回玩家该回合损失的血量。战斗过程由x1的第一个随从作为进攻方开始，接着会从 x2的随从里随机挑选一个作为被攻击方，2个随从均会损失相当于对方攻击力的血量，若血量<=0则随从死亡，即会将该随从所在的节点从链表中移除；接着由x2的第一个随从进攻，接着是x1的第二个随从……直到至少有一方的随从全部死亡为止，仍有随从存活的一方获得本回合胜利，若双方随从全部死亡则平局。由于在战斗阶段的随从死亡并不会使得玩家失去对该随从的所有权，即它仍然可以参加下一场战斗，因此该函数没有使用传递实参地址而是建立临时形参变量，这样对于形参变量的任何删除操作均不会影响实参。

|  |
| --- |
| int Fight(Player x1,Player x2)  {  for (int i = 0; i < x1.h.NumNodes(); i++){  x1.h.Go(i);  x1.h.CurData().LoadBuff();  }  int i = -1,j=-1;  while (x1.h.NumNodes()>0&&x2.h.NumNodes()>0)  {  i++; //当前准备进攻的随从序号  if(i>=x1.h.NumNodes())i=0; //如果之前是最后一个随时进攻则要回到第一个  x1.h.Go(i);  x2.h.Go(rand()%x2.h.NumNodes()); //从2号玩家的随从里随机挑选一个作为被攻击者  Node<Minion> at =\* x1.h.CurNode();  Node<Minion>de=\*x2.h.CurNode(); //判断攻击者攻击力与被攻击者的血量大小关系（攻击者同时也会承受被攻击者的反击）  if (at.GetData().GetATK\_P()>=de.GetData().GetHP\_P())  x2.h.DeleteCurNode(); //大于等于就将被攻击随从移除列表  else  x2.h.CurData().SetHP\_P(de.GetData().GetHP\_P() - at.GetData().GetATK\_P());//小于就改变血量  if (de.GetData().GetATK\_P() >= at.GetData().GetHP\_P())  x1.h.DeleteCurNode(); //反击部分的执行同理  else  x1.h.CurData().SetHP\_P(at.GetData().GetHP\_P()-de.GetData().GetATK\_P());  if (x2.h.NumNodes()==0||x1.h.NumNodes()==0)  break;  }  if(x1.h.NumNodes()==0&&x2.h.NumNodes()==0)return 0; //如果双方都没有随从了就是平局，返回0  if(x1.h.NumNodes()==0)return x2.h.NumNodes(); //如果1号玩家没有随从了就返回一个正值 （2号玩家剩余随从数量），代表一号玩家战败  else  return -x1.h.NumNodes(); //反之返回负值代表2号玩家战败  } |

### 2.2.4全局函数

以下六个全局函数都是用于用户交互界面中的输出。

|  |
| --- |
| void Menu(int c); //总菜单  void Shop(); //显示随从和Buff的所有数据  void Settings(); //展示设置界面  void Thanks(); //展示制作人员名单  void ReSet(); //重置所有设置为初始值  void OP(); //显示开发者选项菜单 |

# **3.main()函数分析**

主函数负责总起调控上述的所有函数，其首先会调用文件读取函数对3个类的静态公有变量进行值的更新操作，文件内的数据由上一次程序结束运行时的文件写入函数将当前的上述变量值写入，由此实现即使程序结束运行用户对游戏设置的更改仍可以保存到下一次打开游戏时。接着会使用 switch结构根据玩家的不同输入调用相对于的显示函数和操作函数完成对应功能。最后会将当前的各个静态公有变量的值写入文件以实现设置的记录。

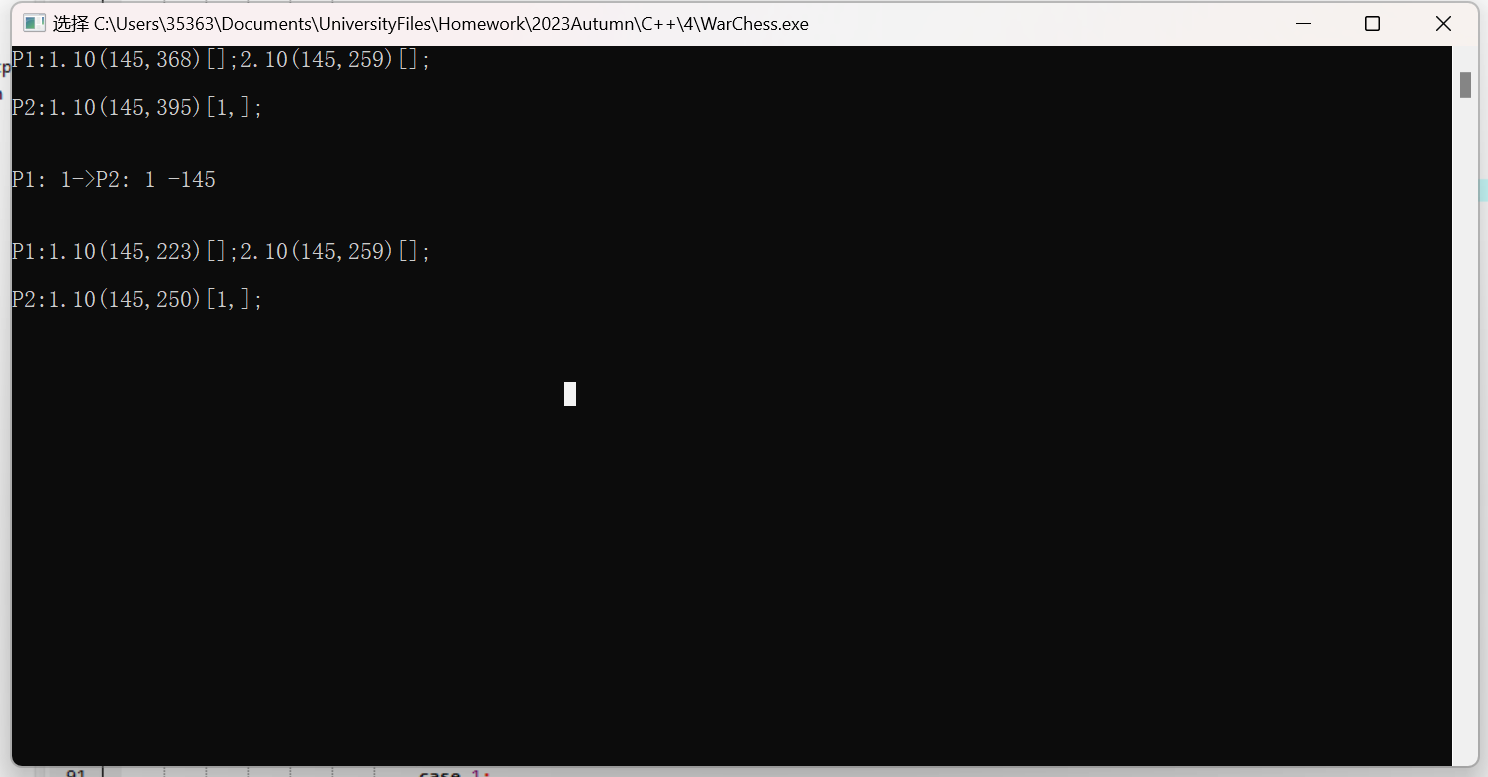
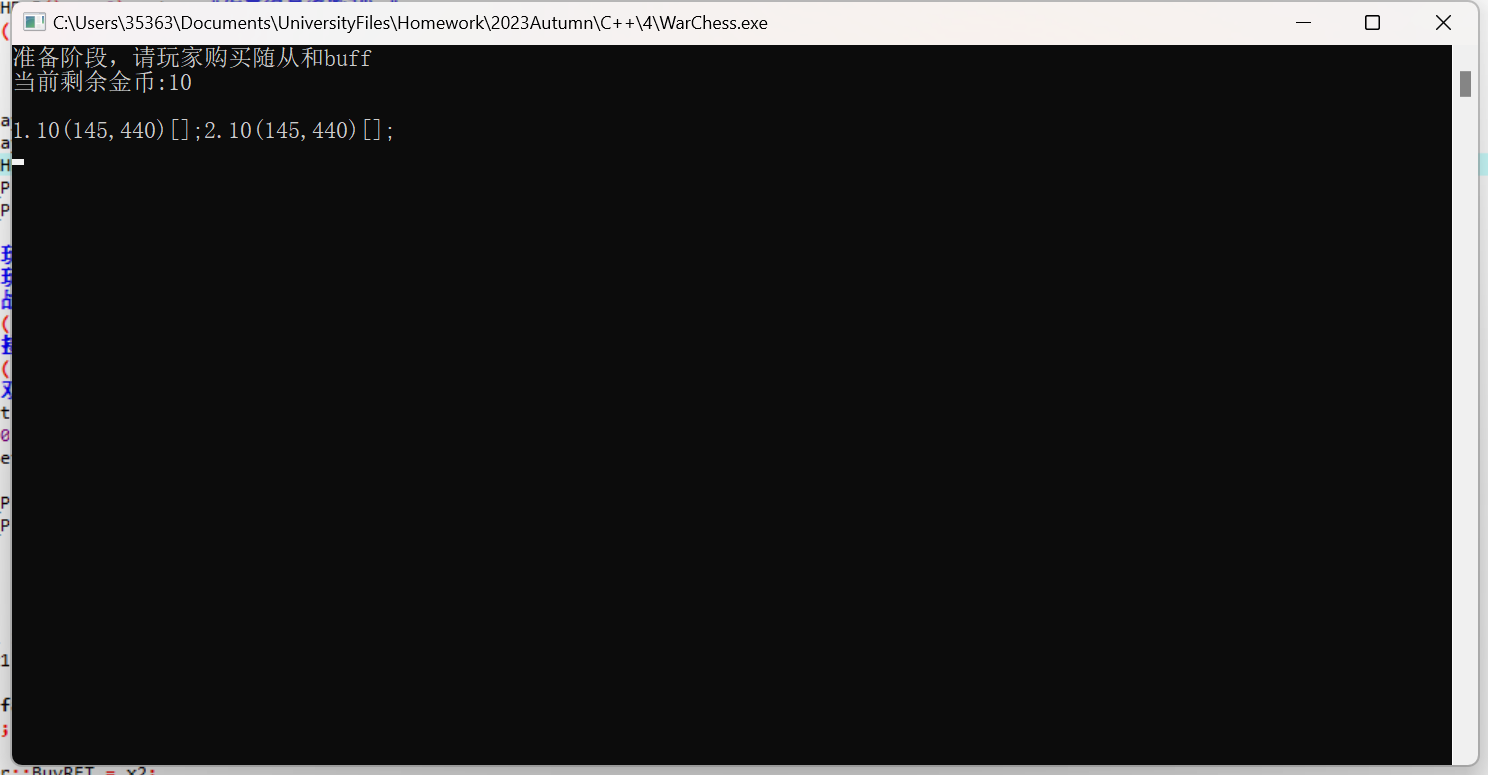
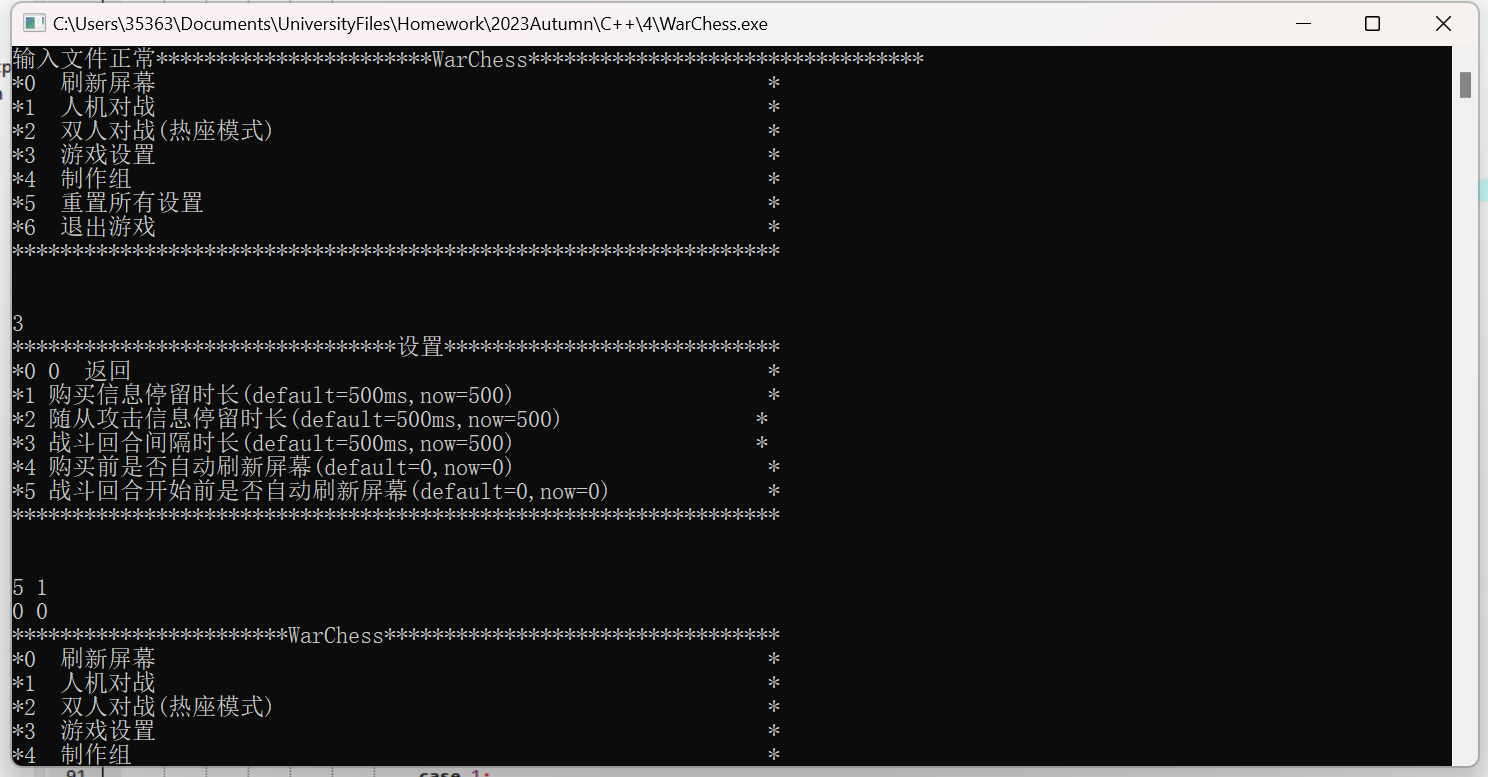
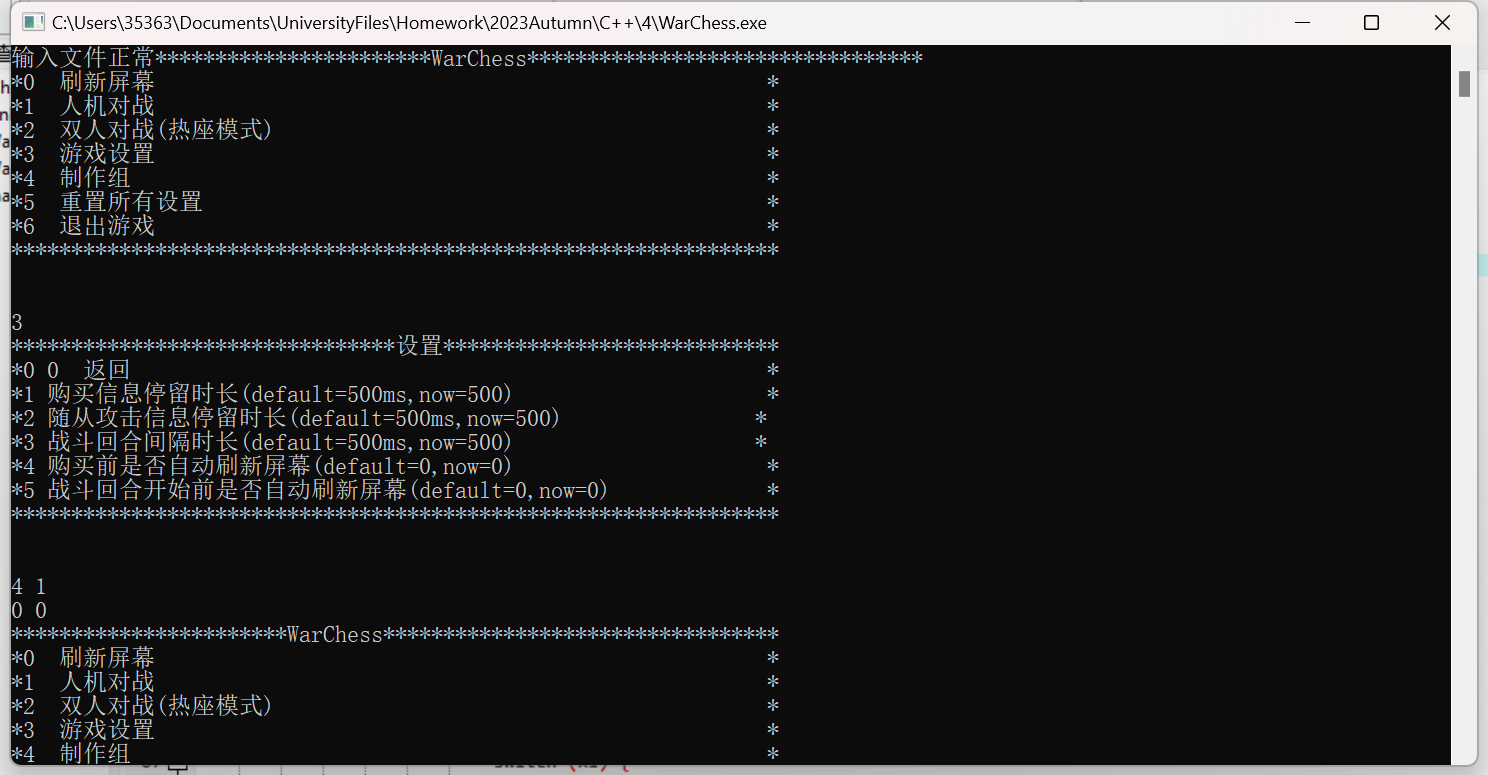
主函数中主要部分是个大的switch语句，各个分支控制整个系统不同功能的实现：具体功能在演示视频内展示。

# **3 测试情况**

## **3.1 测试样例设计**

### 基本功能测试

#### **3.1.1.1“刷新”**



#### **3.1.1.2单机对战**

第一场战斗：

玩家一阵容：1.随从8【buff4】，2.随从5，3.随从6，4.随从6

人机阵容：1.随从1【buff2,buff3,buff3】，2.随从2【buff4,buff2】

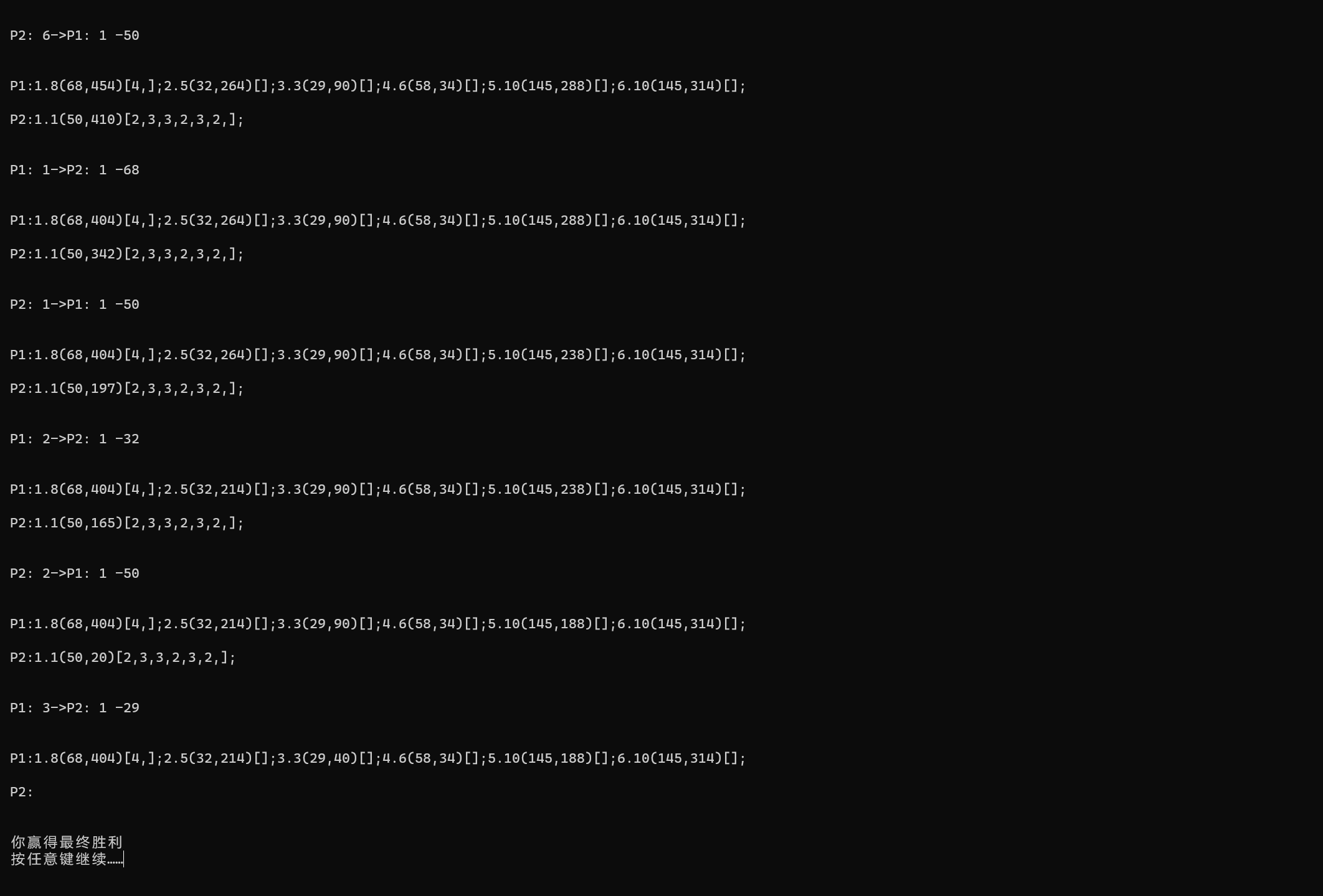
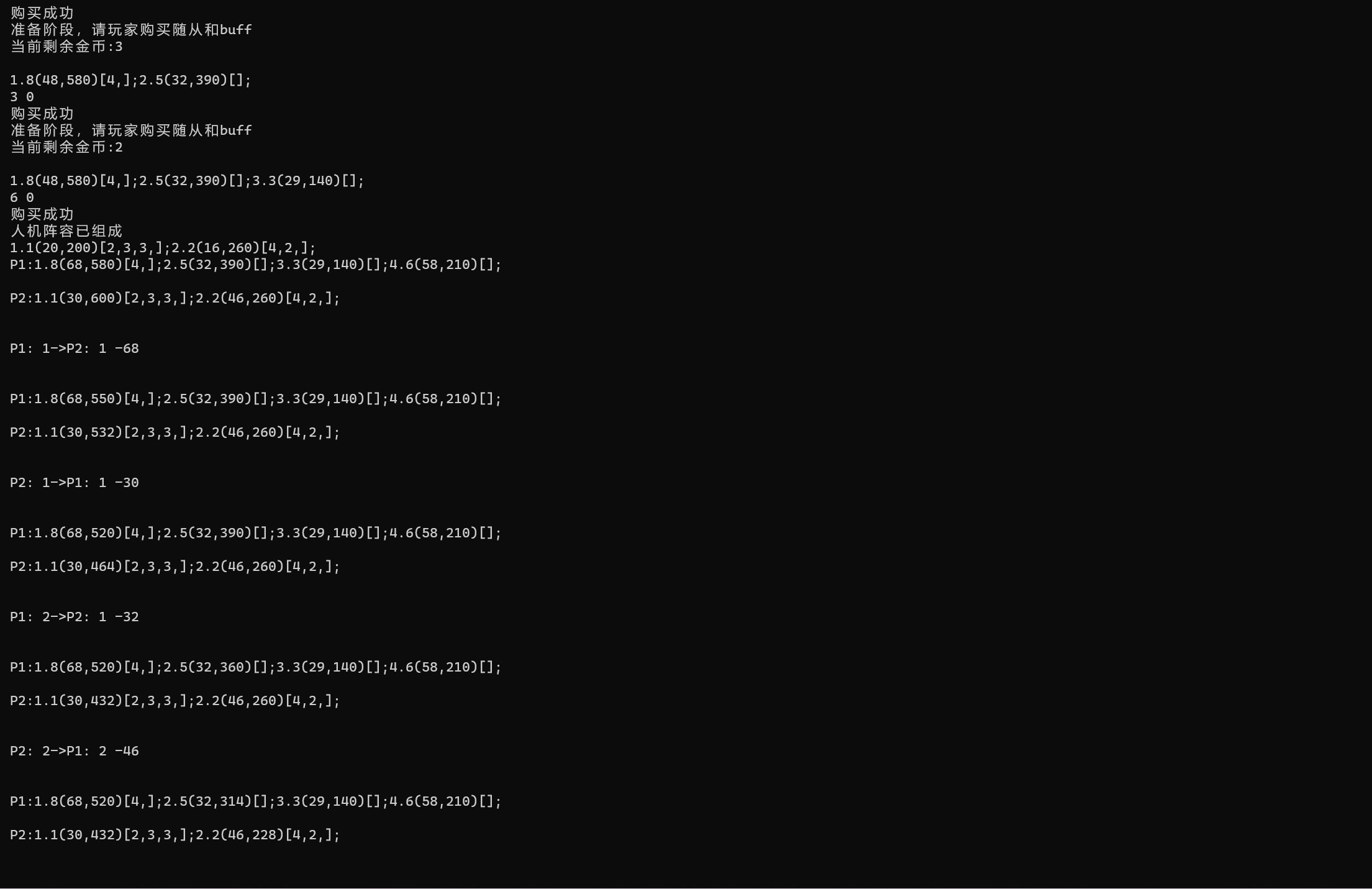
第一场战斗结束后：玩家一血量不变，玩家二（人机）血量减少为1

第二场战斗：

玩家一阵容：1. 随从10，2.随从10

玩家二阵容：1.随从1【buff2,buff3,buff3,buff2,buff,buff3,buff2】，2.随从2【buff4,buff2,buff4,buff1,buff2,buff3】

本轮游戏结束后，玩家获胜



#### **3.1.1.3双人对战**

第一场战斗：

玩家一阵容：1.随从10，2.随从10

玩家二阵容：1.随从6【buff2】，2.随从5【buff3】，3.随从4，4.随从3

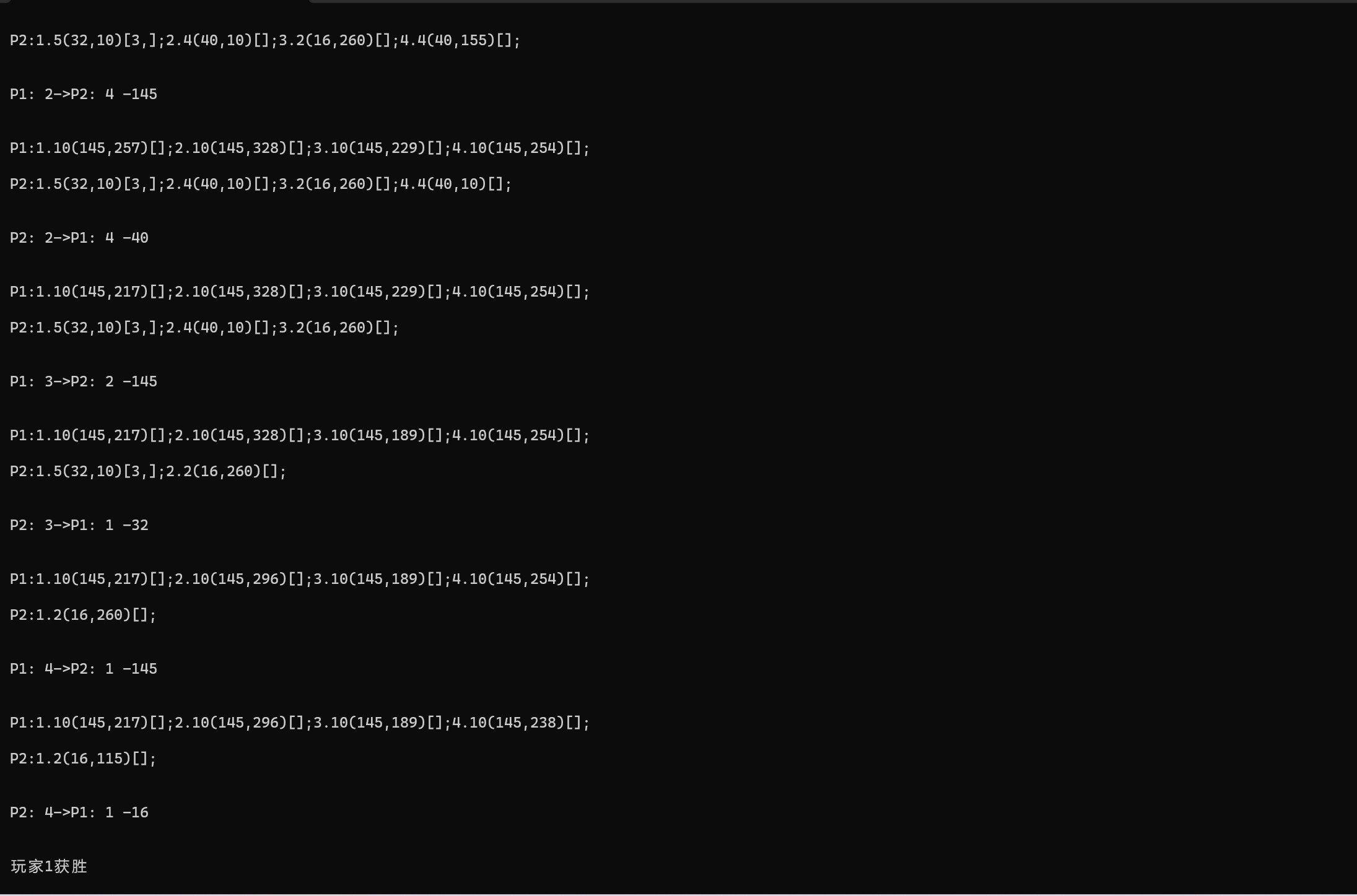
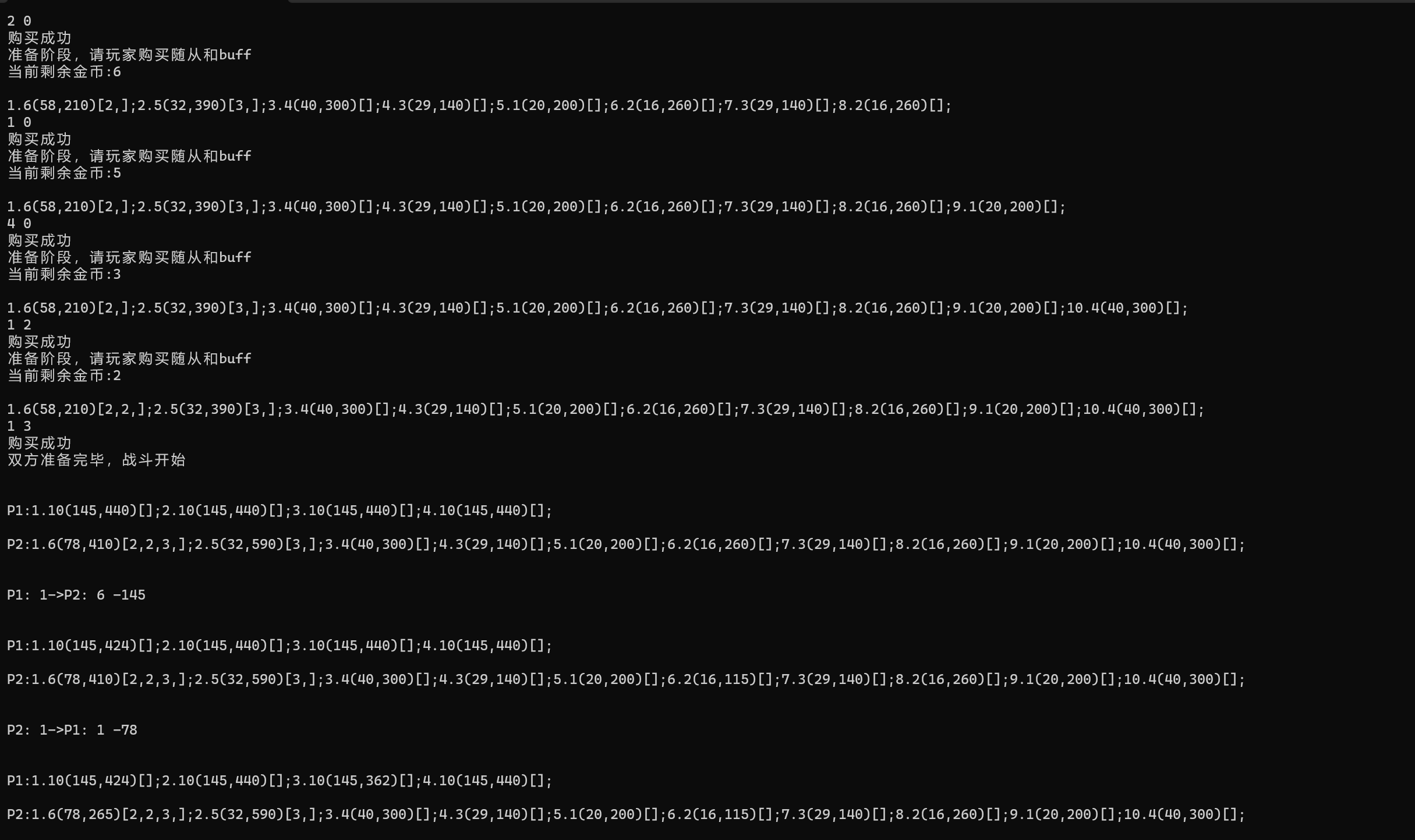
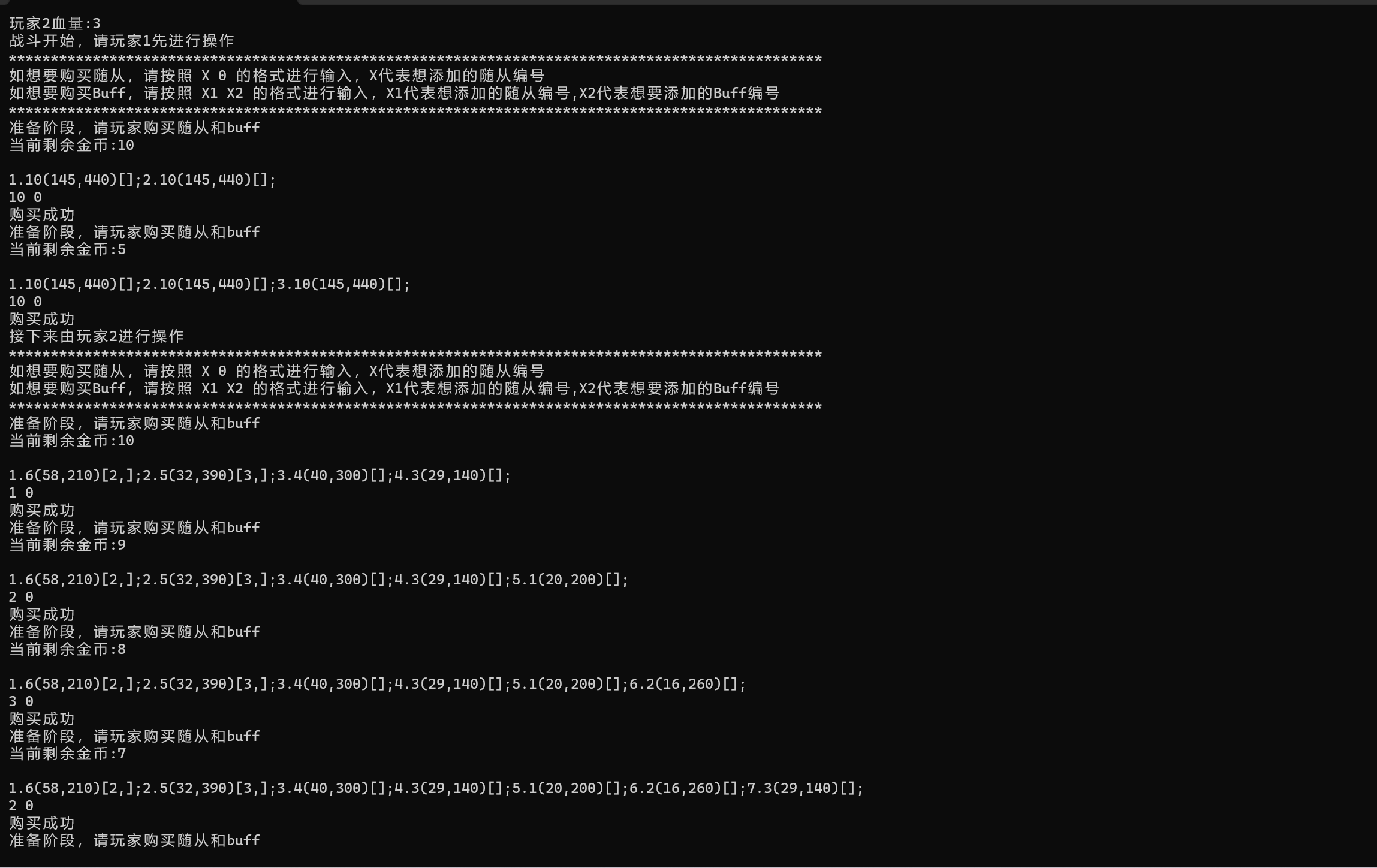
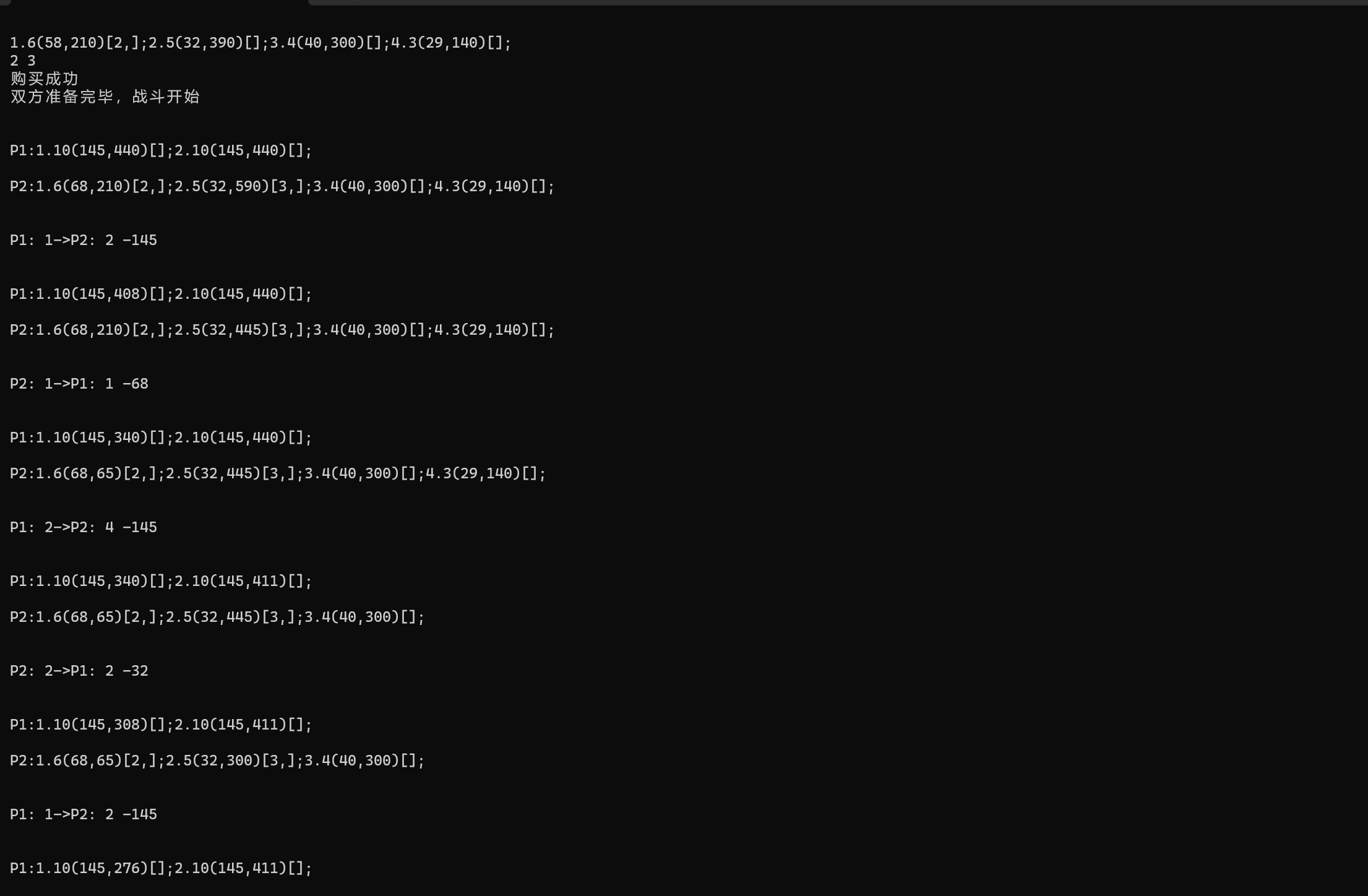
第一场战斗结束后：玩家一血量不变，玩家二血量减少为3

第二场战斗：

玩家一阵容：1. 随从10，2.随从10

玩家二阵容：1.随从1【buff2,buff3】，2.随从2，3.随从3，4.随从2，5.随从5，6.随从1，7.随从4

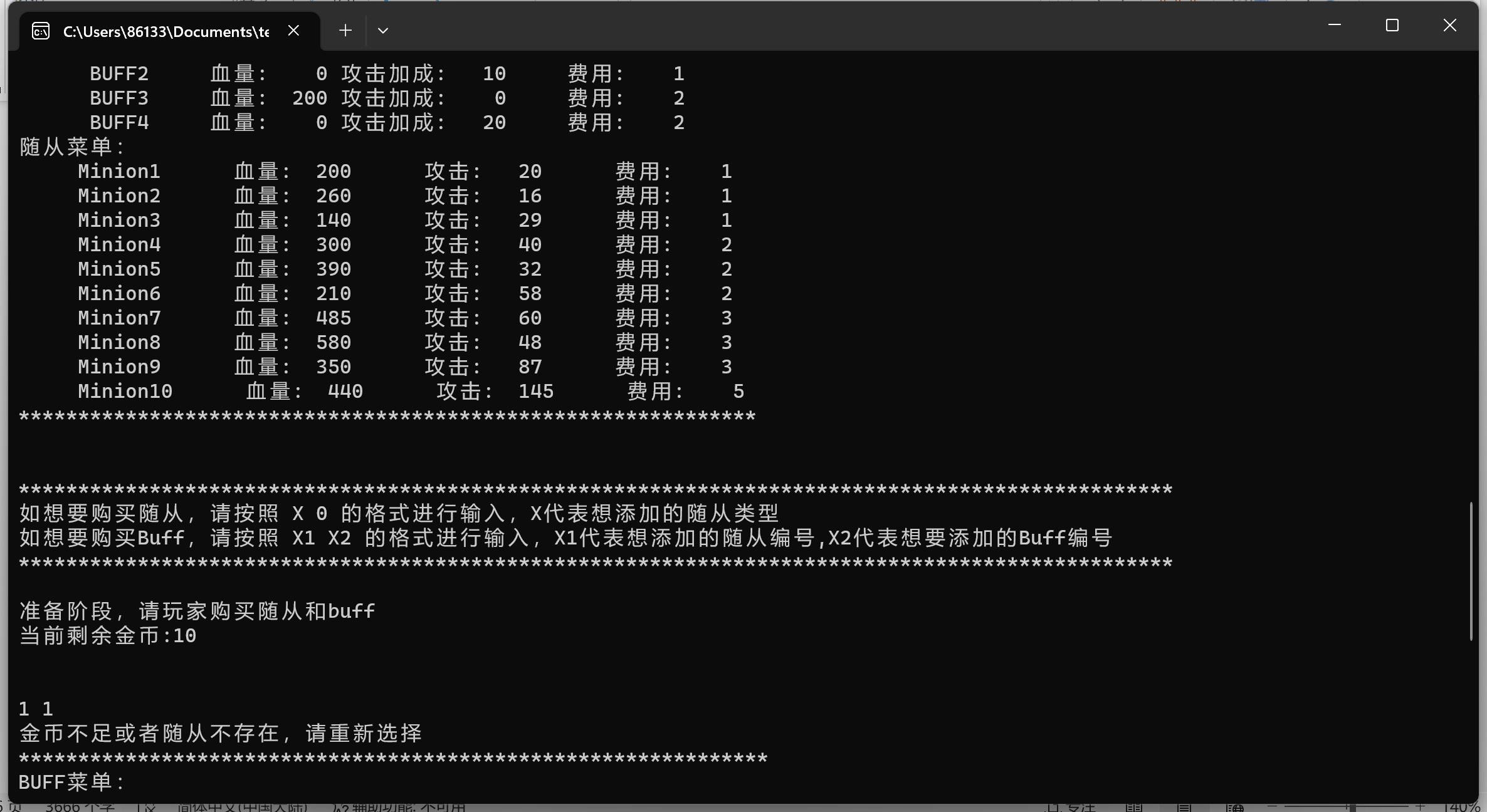
本轮游戏结束后，玩家一获胜



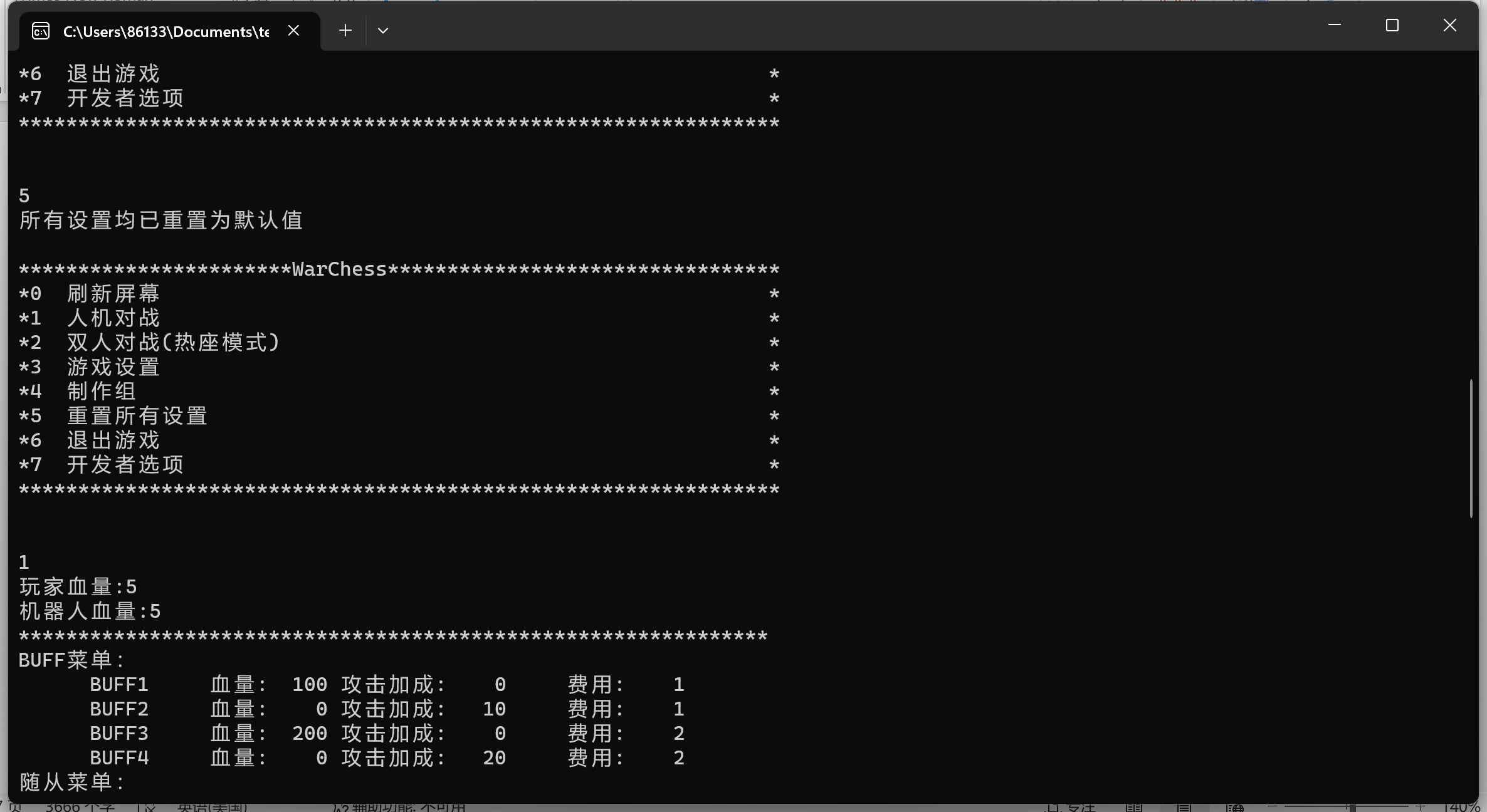
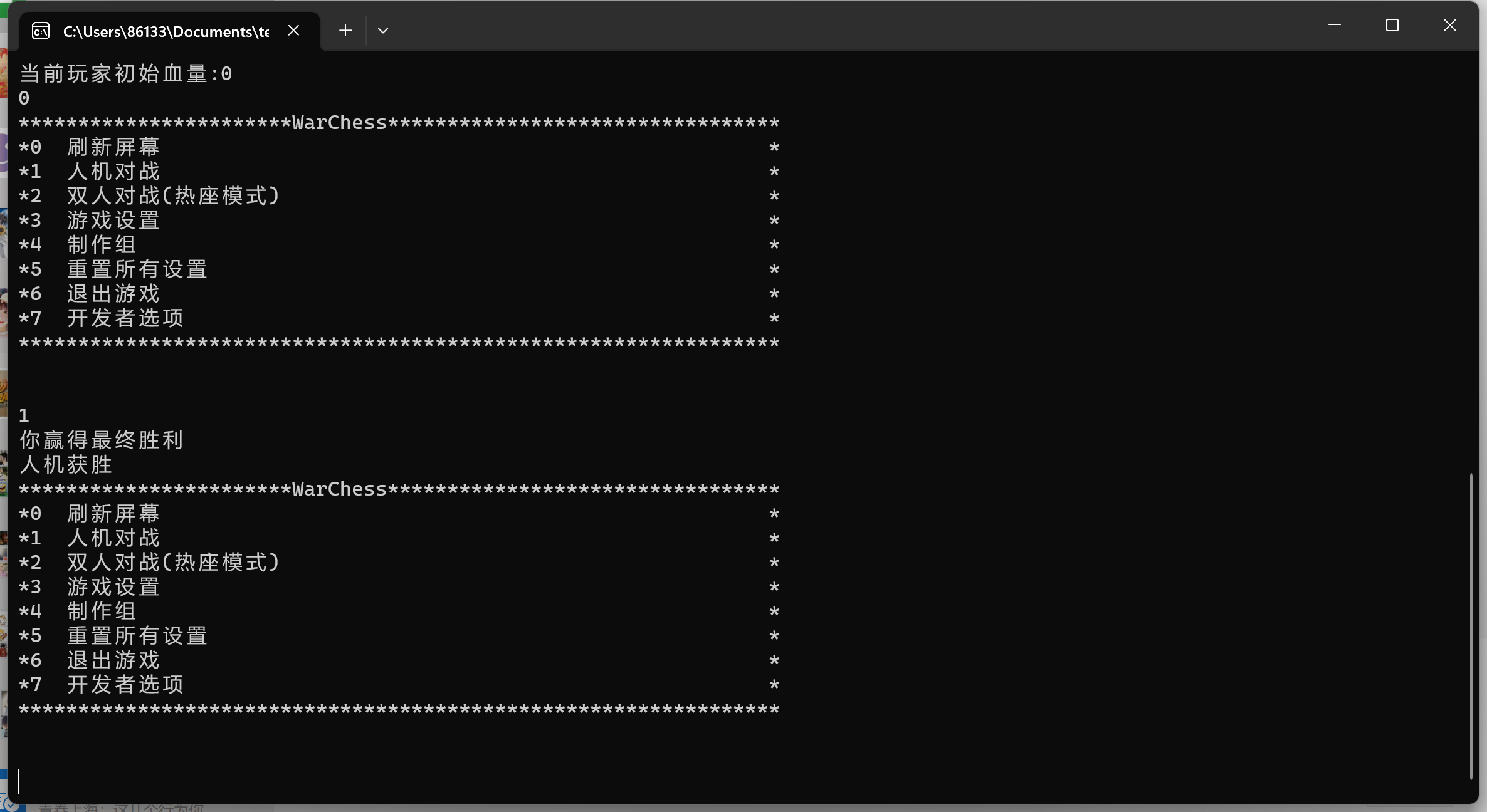
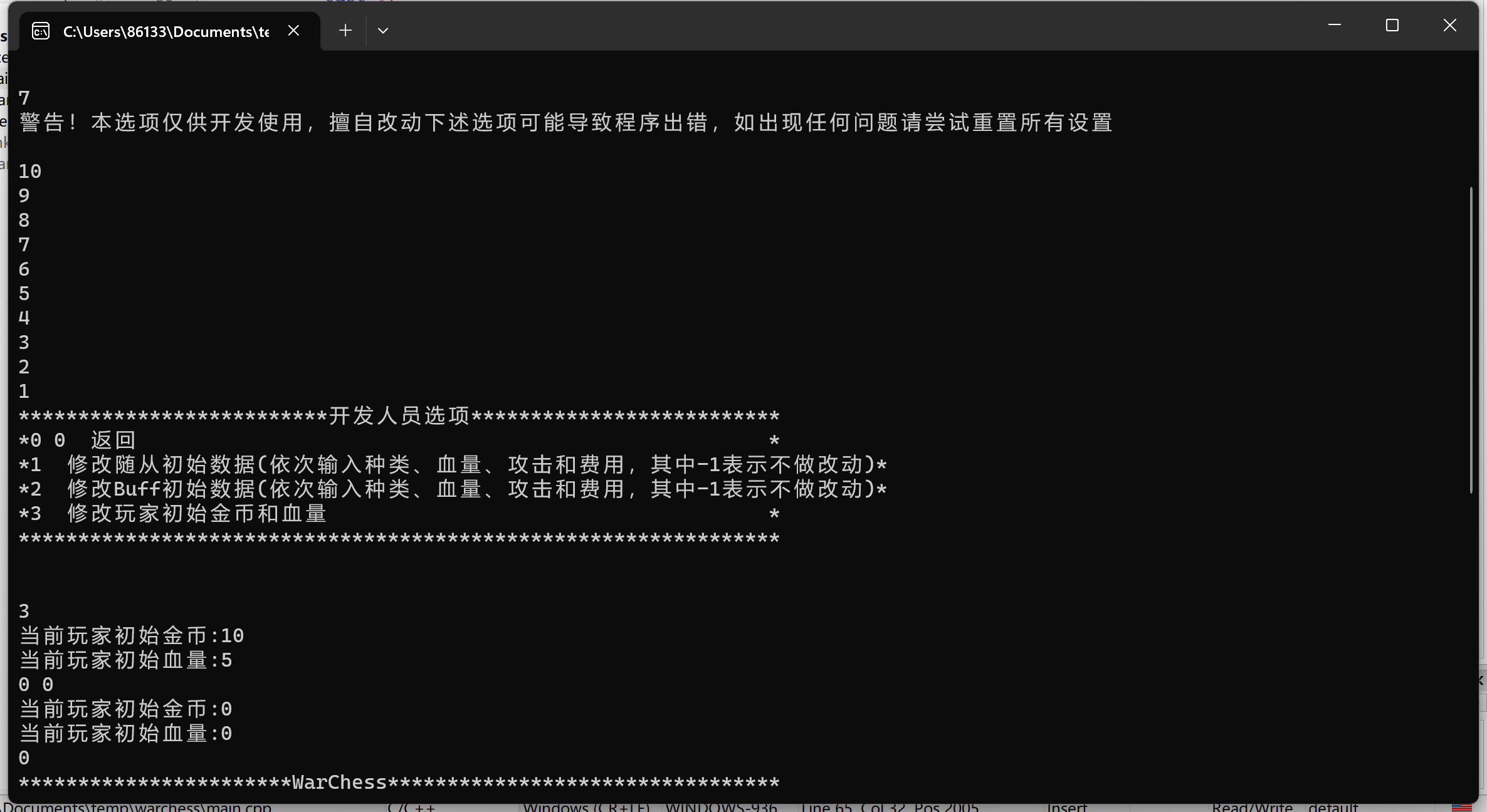
### 3.1.2 可靠性测试

当双人对战时，如果双方中选择的随从只有一种类型（例如，双方输入都为10 0和10 0，此时双方都各有两个相同的10号随从），且没有增加任何属性时，战斗结果将为平局，除了这种极其特殊的情况，由于战斗过程种随机数的处理，平局几乎不可能发生。

在创建阵容时，通过buy函数对输入数据的判断，也很好避免了数组下标越界的问题。



即使在开发者模式下将玩家的初始生命值和金币值设置为了0，程序也不会发生闪退，而是直接退出了战斗，玩家仍然可以通过重置设置使游戏可以正常运行下去。



## **3.2 测试结果对程序的改进情况**

血量为0时仍旧进入战斗回合，修改判定条件。读入文件失败，修改读入文件位置为相对位置。战斗时Buff未添加，增加准备函数。输入时提示不清，重新设计Ui界面，增加提示性语句。迎合不同玩家的需求，增加设置界面及开发者模式，并调整输入输出函数使得玩家自定义数据可以的到保存。