

《.net程序设计》大作业报告

一个基于WPF的文字冒险RPG游戏

学 院 信息科学与技术学院

专 业 计算机科学与技术系

成 员： 崔金泽 20192163 信1901-3

陈顺鹏 20194223 信1901-3

韩英杰 20191264 信1901-1

指导教师： 李 静

完成日期 2021年 11月20 日

**摘 要**

文字冒险游戏，又名电子小说，也被称为交互式小说。以精彩的剧情为卖点的游戏。它以文字叙述为主，并以CG或动画为辅演出剧情，并设有分支和多个结局。此类型游戏以软件模拟情境，令玩家使用文字指令控制角色，以影响周边的情境。其运作方式可以理解成是用电脑游戏来做文学叙事。

Roguelike本身的理念源自于二十世纪七十年代的游戏，包括Adventure (1975)、Dungeon (1975)、DND/Telengard (1976)、 Beneath Apple Manor (1978)，并利用了一些PLATO系统（即第一代网络游戏），完成了基础建设。目的是在电脑上再现“DND”游戏体验，并且严格遵循“DND”游戏规则的单人回合扮演游戏。这款游戏在当时非常受欢迎，以致后来衍生出了一系列与“Rogue”相类似或者同类型的游戏作品，这些作品被统称为“Roguelike”。

C#是一种安全的、稳定的、简单的、优雅的，由C和C++衍生出来的面向对象的编程语言。它在继承C和C++强大功能的同时去掉了一些它们的复杂特性（例如没有宏和模版，不允许多重继承)。C综合了VB简单的可视化操作和C++的高运行效率，以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程的支持成为.NET开发的首选语言。

这是一款类比人生模拟器类游戏的软件，该游戏针对的是喜欢探险类游戏的人群，喜欢刺激的玩家会具备较强的体验感，这个软件开始可以让你身临其境，去体会不同职业的探险乐趣，玩家对每个职业都可以分配各种能力值，如力量值，活力值，敏捷度等，但是总和有上限，玩家对每个职业都可以分配各种能力值，如力量值，活力值，敏捷度等，但是总和有上限，你可以尽情地探索这个世界，而且在游戏探险的过程中，你也可以装备自己的武器，还有头部，身体等部位，让自己变得更加强壮，本软件在C#项目中使用SQLite数据库实现增删改查，窗体美观。

关键词**：**人生模拟器 游戏 c# SQLite

目 录

[一 引言 1](#_Toc90212948)

[1.1 编写目的 1](#_Toc90212949)

[1.2 目的与意义 2](#_Toc90212950)

[1.3 分工 2](#_Toc90212951)

[二 需求分析 2](#_Toc90212952)

[2.1 技术可行性 2](#_Toc90212953)

[2.2 经济可行性 4](#_Toc90212954)

[2.3 社会可行性 4](#_Toc90212955)

[2.4 概念解释 5](#_Toc90212956)

[2.5 玩家的需求 6](#_Toc90212957)

[三 程序概要设计 7](#_Toc90212958)

[3.1程序介绍 7](#_Toc90212959)

[3.2功能分析 8](#_Toc90212960)

[四 系统的总体结构 8](#_Toc90212961)

[五 数据库设计 11](#_Toc90212962)

[5.1数据库表结构 12](#_Toc90212963)

[5.2数据库E-R图 14](#_Toc90212964)

[六 系统实现 16](#_Toc90212965)

[6.1系统具体实现 16](#_Toc90212966)

[6.2项目改进 33](#_Toc90212967)

[六 心得体会 35](#_Toc90212968)

[七 参考文献 35](#_Toc90212969)

[八 附录（核心源代码） 36](#_Toc90212970)

[SQLiteHelper.cs 36](#_Toc90212971)

[Card.cs 40](#_Toc90212972)

[Hand.cs 43](#_Toc90212973)

[Item.cs 46](#_Toc90212974)

# 一 引言

## 1.1 编写目的

随着电脑和网络的普及，人们越来越依赖于电脑。过去人们通过锻炼健身等户外活动来放松和娱乐自己；现在人们更多的是在闲暇时通过玩一盘或几盘轻松娱乐的游戏来达到放松娱乐的目的。

角色扮演游戏（Role-playing game），简称为RPG，是游戏类型的一种。在游戏中，玩家负责扮演这个角色在一个写实或虚构世界中活动。玩家负责扮演一个或多个角色，并在一个结构化规则下通过一些行动令所扮演的角色发展。玩家在这个过程中的成功与失败取决于一个规则或行动方针的形式系统（Formal system）。Roguelike是欧美国家对一类游戏的统称，是角色扮演游戏（RPG）的一个子类（Roguelike-RPG）。指二十世纪八十年代初，由Michael Toy和Glenn Wichman两位软件工程师共同在UNIX系统上开发，在大型机上运行的游戏。

C#是一种安全的、稳定的、简单的、优雅的，由C和C++衍生出来的面向对象的编程语言。它在继承C和C++强大功能的同时去掉了一些它们的复杂特性（例如没有宏和模版，不允许多重继承)。C综合了VB简单的可视化操作和C++的高运行效率，以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程的支持成为.NET开发的首选语言。

并且C#成为ECMA与ISO标准规范。C#看似基于C++写成，但又融入其它语言如Pascal、Java、VB等。

Microsoft在正式的场合把C#描述为一种简单、现代、面向对象、类型非常安全、派生于C和C++的编程语言。大多数独立的评论员对其说法是“派生于C、C++和Java”。这种描述在技术上是非常准确的，但没有涉及到该语言的真正有点。从语法上看，C#非常类似于C和Java,许多关键字都是相同的，C#也使用类似于C和C++的块结构，并用括号()来标识代码块，用分号分隔各行语句。对C#代码的第一印象是它非常类似于C和Java代码。但在这些表面上的类似性后面，C#学习起来要比C++容易得多，但比Java难一些。其设计与现代开发工具的适应性要比其他语言更高，它同时具有Visual Basic的易用性、高性能以及C++的低级内存访问性。

## 1.2 目的与意义

随着现在生活节奏的加快，人们在工作之余需要一个轻松娱乐的放松环境。因此我编写此软件的目的，一方面作为对我大学三年的学习成果的一次检验，另一方面开发此软件使用户能在娱乐休闲的同时获得快乐，游戏难度设计的并不简单，需要玩家仔细思考，不过当玩家通关之后，获得的成就感是不言自明的。

## 1.3 分工

|  |  |  |  |
| --- | --- | --- | --- |
| 信1901-3 | 20192163 | 崔金泽 | 界面设计+游戏数值设计，平衡设计+报告整理 |
| 信1901-3 | 20194223 | 陈顺鹏 | 逻辑设计+主要的代码编辑 |
| 信1901-1 | 20191264 | 韩英杰 | 文本编辑+调试，整合程序 |

# 二 需求分析

## 2.1 技术可行性

利用VS2019开发的仿windos自带益智游戏开发的冲关益智游戏，其中大量运用了循环技术，界面主要是使用了button控件与label控件来实现的，还有采用定义新类来生成分数。在结合大量的循环方法来实现整个软件，利用循环来生成总分并且在窗体上显示，利用循环来实现在表格上随机增加，并且完成鼠标单击时间的功能。C#是微软公司发布的一种面向对象的、运行于.NET Framework之上的高级程序设计语言。并定于在微软职业开发者论坛(PDC)上登台亮相。C#看起来与Java有着惊人的相似；它包括了诸如单一继承、接口、与Java几乎同样的语法和编译成中间代码再运行的过程。但是C#与Java有着明显的不同，它借鉴了Delphi的一个特点，与COM（组件对象模型）是直接集成的，而且它是微软公司 .NET windows网络框架的主角。

开发工具：

Visual Studio 是一套完整的开发工具集，用于生成 http://ASP.NET Web 应用程序、XML Web Services、桌面应用程序和移动应用程序。Visual Basic、Visual C++、Visual C# 和 Visual J# 全都使用相同的集成开发环境 (IDE)，利用此 IDE 可以共享工具且有助于创建混合语言解决方案。另外，这些语言利用了 .NET Framework 的功能，通过此框架可使用简化 ASP Web 应用程序和 XML Web Services 开发的关键技术。

功能如下：

　创建满足关键性要求的多层次的智能客户端、Web、移动或基于Microsoft Office的应用程序。使用Visual Studio 2019, 专业开发人员能够：

　　使用改进后的可视化设计工具、编程语言和代码编辑器，享受高效率的开发环境

　　在统一的开发环境中，开发并调试多层次的服务器应用程序。

　　使用集成的可视化数据库设计和报告工具，创建SQL Server 2008解决方案。

　　使用Visual Studio SDK创建可以扩展Visual Studio IDE的工具。

　　Microsoft为单独工作或在小型团队中的专业开发人员提供了两种选择，Visual Studio 2008 Professional Edition和用于Microsoft Office系统的Visual Studio 2008工具。每种版本都在标准版的特性上进行了扩展，包括用于远程服务程序开发和调试、SQL Server2008开发的工具，以及完整的、没有限制的开发环境。每种产品都可以单独购买或打包定购。

专业开发人员喜欢自由的使用.NET Framework 2.0，它是一种稳健的、功能齐备的开发环境，支持创建扩展Visual Studio集成开发环境的工具。

主要部分：

公共语言运行库：运行库实际上在组件的运行时和开发时操作中都起到很大的作用，尽管名称中没有体现这个意思。在组件运行时，运行库除了负责满足此组件在其他组件上可能具有的依赖项外，还负责管理内存分配、启动和停止线程和进程，以及强制执行安全策略。在开发时，运行库的作用稍有变化；由于做了大量的自动处理工作（如内存管理），运行库使开发人员的操作非常简单，尤其是与今天的 COM 相比。特别是反射等功能显著减少了开发人员为将业务逻辑转变为可重用组件而必须编写的代码量。统一编程类：该框架为开发人员提供了统一的、面向对象的、分层的和可扩展的类库集 (API)。目前，C++ 开发人员使用 Microsoft 基础类，而 Java 开发人员使用 Windows 基础类。该框架统一了这些完全不同的模型，还为 Visual Basic 和 JScript 程序员提供了对类库的访问。通过创建跨所有编程语言的公共 API 集，公共语言运行库使得跨语言继承、错误处理和调试成为可能。从 JScript 到 C++ 的所有编程语言具有对框架的相似访问，开发人员可以自由选择它们要使用的语言。

http://ASP.NET：http://ASP.NET 建立在 .NET Framework 的编程类的基础上，为 Web 应用程序模型提供了一组可简化 Web 应用程序生成的控件和基础结构。http://ASP.NET 包括可用于封装通用 HTML 用户界面元素（如文本框、按钮和列表框）的一组控件。但这些控件在 Web 服务器上运行，并以 HTML 的形式将其用户界面呈现在浏览器中。在服务器上，这些控件公布面向对象的编程模型，该模型为 Web 开发人员提供面向对象编程的丰富功能。http://ASP.NET 还提供基础结构服务，如状态管理和进程回收，从而可以进一步减少开发人员必须编写的代码数量，并提高应用程序的可靠性。另外，http://ASP.NET 使用这些同样的概念使开发人员能够以服务的形式交付软件。使用 XML Web Services 功，http://ASP.NET 开发人员可以编写他们的业务逻辑，并使用 http://ASP.NET 基础结构通过 SOAP 交付该服务。有关更多信息，请参见使用托管代码进行 XML Web services 编程简介。

## 2.2 经济可行性

基于C#程序设计的文字冒险游戏，从经济角度上来说。该游戏结构简单，容易开发，易于维护，开发时用到的人力和物力相对来说比较少，而且对现在市场上的需求性很高，相对来说也比较实用，对疲惫的各种人群都有很大的帮助。因此，会有一定的市场潜力，也会有不错的经济效益。

## 2.3 社会可行性

根据《中华人民共和国计算机软件保护条例》（1991年6月4日中华人民共和国国务院令第84号发布） （修改后的新条例2002年1月1日起施行）第九条可知：软件著作权人享有下列各项权利:

(一) 发表权, 即决定软件是否公之于众的权利;

(二) 开发者身份权, 即表明开发者身份的权利以及在其软件上署名的权利;

(三) 使用权, 即在不损害社会公共利益的前提下, 以复制、展示、发行、修改、翻译、注释等方式使用其软件的权利;

(四) 使用许可权和获得报酬权, 即许可他人以本条第(三) 项中规定的部分或者全部方式使用其软件的权利和由此而获得报酬的权利;

(五) 转让权, 即向他人转让由本条第(三) 项和第(四) 项规定的使用权和使用许可权的权利。

第十二条 受他人委托开发的软件, 其著作权的归属由委托者与受委托者签订书面协议约定, 如无书面协议或者在协议中未作明确约定, 其著作权属于受委托者。

第十八条 在软件著作权的保护期内, 软件的著作权人或者其受让者有权许可他人行使本条例第九条第(三) 项规定的使用权。著作权人或者其受让者许可他人行使使用权时, 可以按协议收取费用。

另外，软件权利的使用许可应当根据我国有关法规以签订、执行书面合同的方式进行。被许可人应当在合同规定的方式、条件、范围和时间内行使使用权。合同中未明确规定为独占许可的, 被许可的软件权利应当视为非独占的。

综上所述，基于C#程序设计的文字冒险游戏开发在市场上、需求性很大，技术可行。因而决定开发文字冒险游戏。

## 2.4 概念解释

Roguelike类型游戏在2008年的国际Roguelike发展会议上进行了明确的定义，并被命名为“柏林诠释”。在命名并定义的过程中，其中出现频率较高的特点包括：

1、生成随机性。每一次新开局游戏都会随机生成游戏场景，敌人，宝物等不同事物。而玩家的每一次冒险历程也都将是独一无二，不可复制的。更有甚者，只要玩家愿意的话，可以让玩家连扮演的角色都设置成随机生成的。例如敌人，武器，甚至只是一个角色的皮肤或者场景中的河流等。

2、进程单向性。当你在玩一款Roguelike游戏时，存档功能的唯一作用就是记录你当前的游戏进度，每当存档被读取时，对应的进度就会被清空，直到你进行下一次存档。这种存档机制确保玩家无法利用"S/L大法"来降低游戏难度。更甚至在Roguelike玩家圈中，手动备份存档会被当做一种作弊行为受到鄙视。

3、不可挽回性。在大多数Roguelike游戏中，每一个角色只有一次生命，一个角色的死亡意味着玩家将永远失去该角色。无论你是主角，敌人，物品或者场景。甚至在很多Roguelike玩家眼中，Roguelike的最大魅力之一就在于体验每一次不同的死亡与失败，并最大努力的做好当下的自己。

4、游戏非线性。严谨而不失灵活性的游戏规则，使Roguelike具备了极高的自由度，在游戏中，玩家可以发挥想象力，利用各种手段去达成任何他们想做的事情，或合乎常理，或匪夷所思，目的只在于解决他们在游戏中遇到的问题。

5、画面朴素性。其实Roguelike类游戏的鼻祖Rogue被普遍认为是最先有图形界面的冒险游戏之一，虽然这种“图形界面”抽象的过分。但是很多Roguelike类游戏出于对Rogue的致敬，会直接使用ASCII字符来表示游戏画面，更导致很多游戏的主角都叫同一个名字——@。

6、系统复杂性。Rogue本身的复杂程度就远远超过同时期的任何一款作品。而Roguelike类游戏可能会在一款游戏中包括多到无法估量的元素，例如地质、气候和生物分布，以及精细到皮肤、肌肉、血液、骨骼和脂肪的战斗系统，甚至战损痊愈后会留下伤疤以及后遗症。在有些游戏里则可能包括数百种的死亡原因，数千种的生物，数万种的物品。可以说除了Roguelike类游戏，再也没有任何传统游戏能够不计成本的拥有如此多的元素了。

SQLite：

SQLite，是一款轻型的数据库，是遵守ACID的关系型数据库管理系统，它包含在一个相对小的C库中。它是D.RichardHipp建立的公有领域项目。它的设计目标是嵌入式的，而且已经在很多嵌入式产品中使用了它，它占用资源非常的低，在嵌入式设备中，可能只需要几百K的内存就够了。它能够支持Windows/Linux/Unix等等主流的操作系统，同时能够跟很多程序语言相结合，比如 Tcl、C#、PHP、Java等，还有ODBC接口，同样比起Mysql、PostgreSQL这两款开源的世界著名数据库管理系统来讲，它的处理速度比他们都快。SQLite第一个Alpha版本诞生于2000年5月。至2021年已经接近有21个年头，SQLite也迎来了一个版本 SQLite 3已经发布。不像常见的客户-服务器范例，SQLite引擎不是个程序与之通信的独立进程，而是连接到程序中成为它的一个主要部分。所以主要的通信协议是在编程语言内的直接API调用。这在消耗总量、延迟时间和整体简单性上有积极的作用。整个数据库(定义、表、索引和数据本身)都在宿主主机上存储在一个单一的文件中。它的简单的设计是通过在开始一个事务的时候锁定整个数据文件而完成的。

## 2.5 玩家的需求

本游戏用户可以自己练习和娱乐。本游戏需要满足以下几点要求:

(1)利用鼠标点击运行程序。

(2)游戏运行稳定。

(3)遇到bug会有存档，不能因为非用户行为使用户受损

(4)用户可以根据需要暂停或播放背景音乐，并且修改游戏背景和文字大小。

# 三 程序概要设计

## 3.1程序介绍

你可以尽情地探索这个世界，去酒馆市场，银行，教堂等不同的地点接受任务，在每个地点，你可能会遇到各种各样的挑战，此时你可以选择攻击或者逃走，还可以去施展法术，选择咒语，在战斗的过程中，你可以查看敌人的信息，另外，你会有自己的性格，随着战斗的不断深入，你的经验和金币都会增加在游戏探险的过程中，你也可以装备自己的武器，还有头部，身体等部位，让自己变得更加强壮

本游戏的大致流程如下：

1.开始游戏/继续游戏

2.玩家进入初始界面后，可以选择英雄名称，密码，登录，新建英雄等选项

3.当你登录这个页面以后，你就可以尽情的探索了

4.结束游戏，当你选择退出的时候，会弹出退出界面，退出这个程序

游戏的界面如下：

1. 游戏开始界面，作为游戏开始的入口。

2. 游戏主界面，是每个回合玩家选择阶段，与玩家交互的界面。

3. 文案界面，当玩家提交计划之后，可以显示不同的文案。

4. 帮助界面，作为一个独立的界面，可以生成帮助，显示游戏的帮助内容。

5. 设置界面，进行一定的选项设置（如界面主题等）。

## 3.2功能分析

1．你可以选择各种职业，如战士，巫师等

2．游戏需要玩家自己选择喜欢的职业，每个职业都有自己的特点

3．玩家要先设置英雄名称，以及这个英雄对应的密码，确保自己的账号安全

4．玩家对每个职业都可以分配各种能力值，如力量值，活力值，敏捷度等，但是总和有上限

5．玩家在游戏中根据剧情可以选择地点，装备，战斗信息等

大概要实现的有：

1.首先生成一段情景，告诉你现在的信息，如地点，周围环境，之后你可以选择你要去往的地方，或是退出

2.在游戏过程中，你需要安排自己的库存信息，也就是自己的武器装备，如戒指，药水，食物等，需要设计相应的触发条件

3.你还可以中途去市场，里面有武器店，军械库等地方，你还可以在这里购置你的装备，需要对装备信息进行初始化并设计对应的方法

# 四 系统的总体结构



每个职业所佩戴的设备(装备)



每个角色的所具有的特征



管理用户方面具有的功能



系统在软件管理方面的结构



商店的细化



草场的细化



林地的细化



银行功能的设计

# 五 数据库设计

SQLite是一个进程内的轻量级嵌入式数据库，它的数据库就是一个文件，实现了自给自足、无服务器、零配置的、事务性的SQL数据库引擎。它是一个零配置的数据库，这就体现出来SQLite与其他数据库的最大的区别：SQLite不需要在系统中配置，直接可以使用。且SQLite不是一个独立的进程，可以按应用程序需求进行静态或动态连接。SQLite可直接访问其存储文件。

SQLite对于其他数据库的优势：

1.不需要配置，不需要安装和管理。

2.不需要一个单独的服务器进程或操作的系统（无服务器的）。

3.一个完整的SQLite数据库存储在一个单一的跨平台的磁盘文件上。

4.SQLite是非常小的，轻量级的数据库，完全配置时小于400KiB，省略可选功能配置时小于250KB。

5.SQLite是一个自给自足的数据库，这也就意味着不需要任何外部的依赖。

6.SQLite事务是完全兼容ACID的，允许从多个进程或线程安全访问。

7.SQLite支持SQL92（SQL2）标准的大多数查询语言的功能。

8.SQLite使用ANSI-C编写的，并提供了简单和易于使用的API。

9.SQLite可在UINX（Linux，Max OS-X,Android，iOS）和Windows（Win32，WinCE，WinRT）中运行。

## 5.1数据库表结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 角色 | | | | |
| 字段名 | 数据类型 | 长度 | 是否为NULL | 描述 |
| YHID | INT | 8 | 不为NULL | 创建该角色的ID |
| JSID | INT | 8 | 不为NULL | 该角色ID |
| JSNC | INT | 8 | 不为NULL | 角色昵称 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 角色属性 | | | | |
| 字段名 | 数据类型 | 长度 | 是否为NULL | 描述 |
| ID | INT | 8 | 不为NULL | 拥有该属性的角色ID |
| LL | INT | 3 | 不为NULL | 力量 |
| ZH | INT | 3 | 不为NULL | 智力 |
| MJ | INT | 3 | 不为NULL | 敏捷 |
| ZY | INT | 3 | 不为NULL | 经验值 |
| JYZ | INT | 3 | 不为NULL | 职业 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 通用属性 | | | | |
| 字段名 | 数据类型 | 长度 | 是否为NULL | 描述 |
| SMZ | INT | 3 | 不为NULL | 生命值 |
| MFZ | INT | 3 | 不为NULL | 魔法值 |
| GJL | INT | 3 | 不为NULL | 攻击值 |
| HJZ | INT | 3 | 不为NULL | 护甲值 |
| MKZ | INT | 3 | 不为NULL | 魔抗值 |
| ID | INT | 8 | 不为NULL，外键 | 拥有该属性的角色ID |

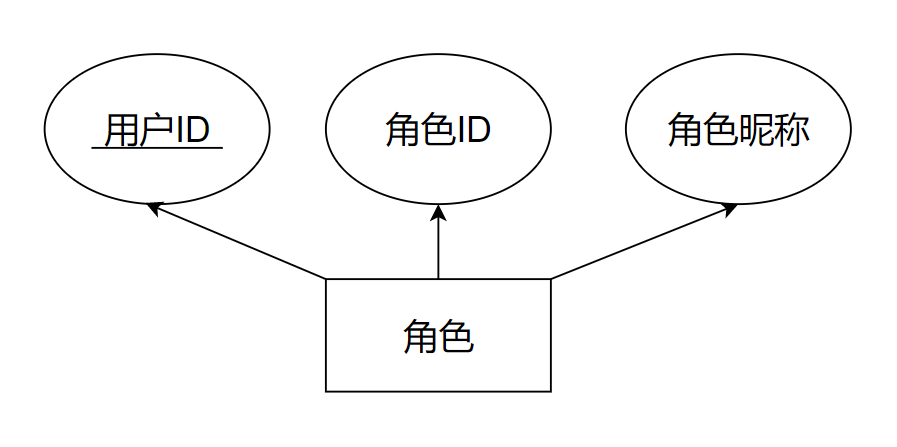
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用户信息 | | | | |
| 字段名 | 数据类型 | 长度 | 是否为NULL | 描述 |
| YHID | INT | 8 | 不为NULL | 用户ID |
| YHMM | CHAR | 8 | 不为NULL | 用户密码 |
| DLDZ | CHAR | 8 | 可NULL | 登录地址 |
| DLSJ | CHAR | 8 | 可NULL | 登录时间 |

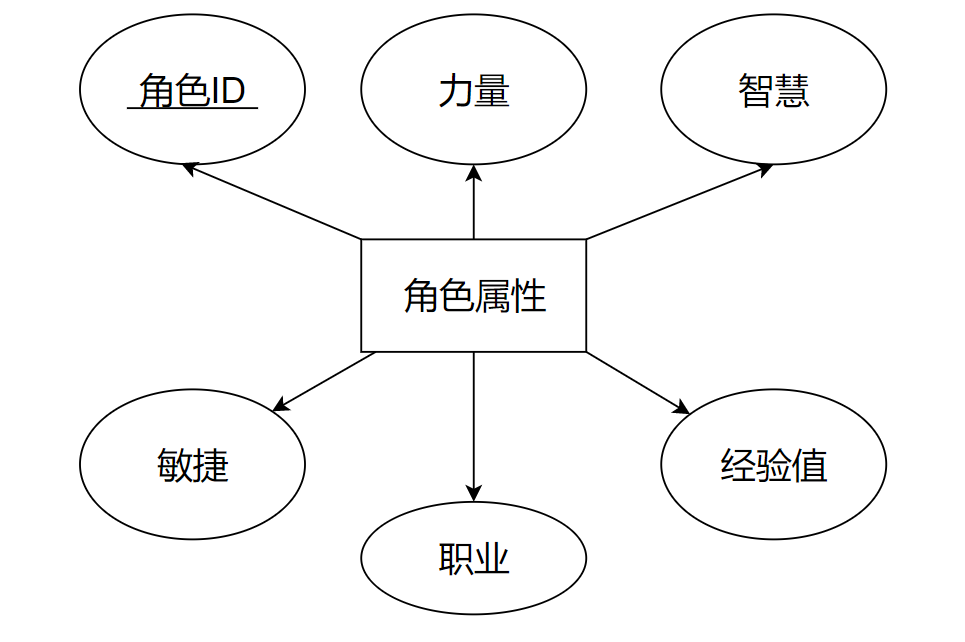
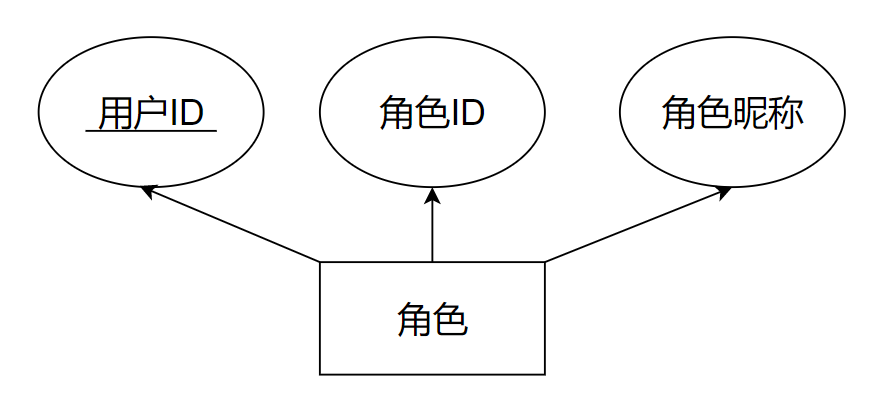
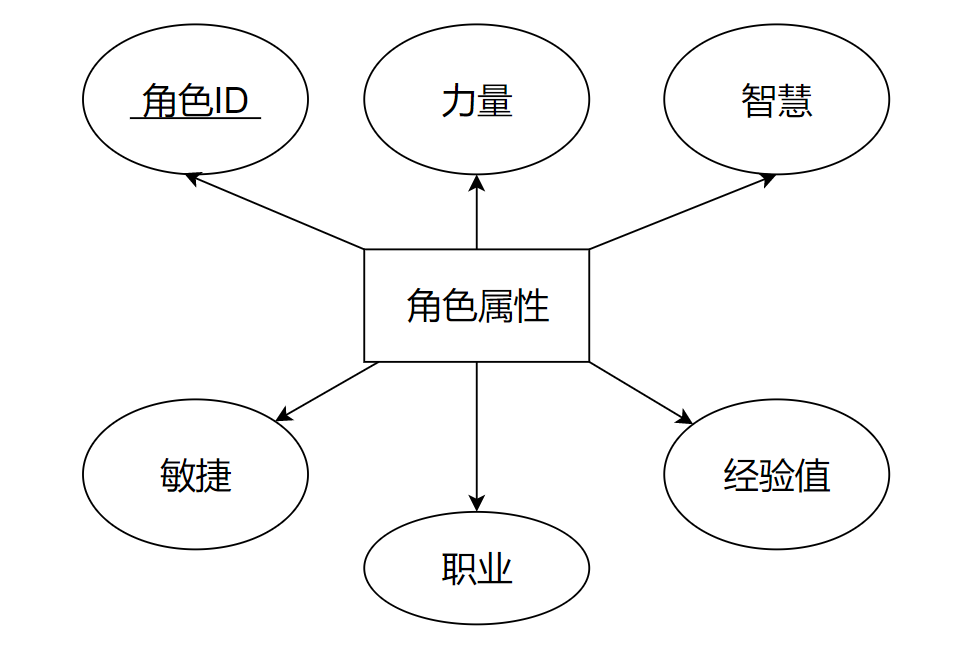
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 装备 | | | | |
| 字段名 | 数据类型 | 长度 | 是否为NULL | 描述 |
| ZBID | INT | 8 | 不为NULL，主键 | 装备ID |
| ZRID | INT | 8 | 不为NULL，外键 | 拥有该装备的角色ID |
| ZBMC | CHAR | 8 | 不为NULL | 装备名称 |
| ZBMS | CHAR | 8 | 可NULL | 装备描述 |

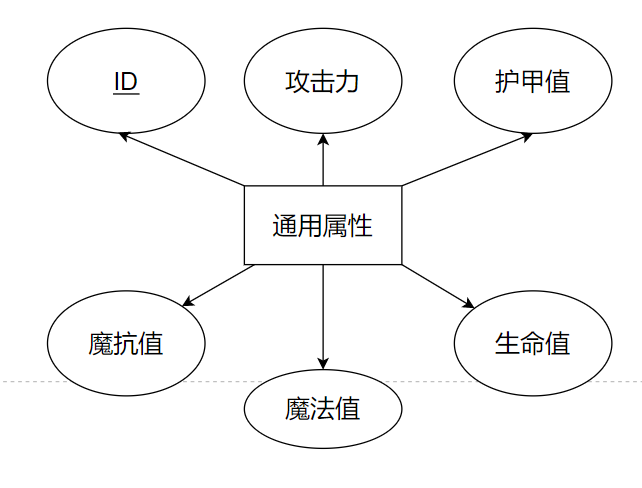
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NPC属性 | | | | |
| 字段名 | 数据类型 | 长度 | 是否为NULL | 描述 |
| ID | INT | 8 | 不为NULL，外键 | 拥有该属性的ID |
| CSDD | CHAR | 8 | 不为NULL | 出生地点 |
| HHFF | INT | 3 | 不为NULL | 活动范围 |
| GN | CHAR | 3 | 可NULL | 功能 |

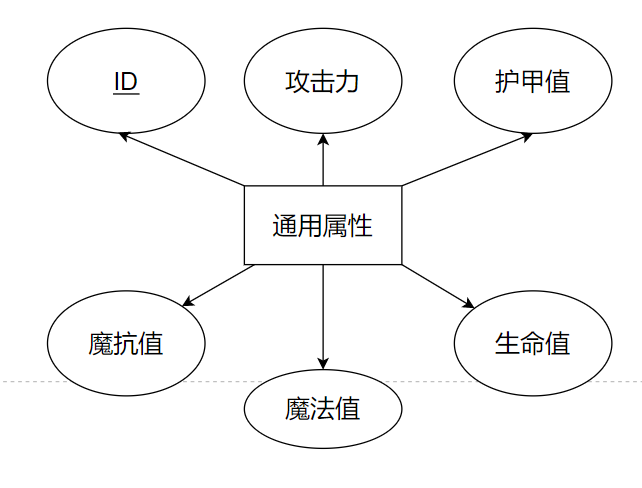
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 角色属性 | | | | |
| 字段名 | 数据类型 | 长度 | 是否为NULL | 描述 |
| ID | INT | 8 | 不为NULL | 拥有该技能的角色ID |
| JNMC | CHAR | 3 | 不为NULL | 技能名称 |
| JCSS | INT | 3 | 不为NULL | 基础伤害 |
| JCLX | CHAR | 3 | 不为NULL | 加成类型 |
| JCBL | INT | 3 | 不为NULL | 加成比例 |
| LQSJ | INT | 3 | 不为NULL | 冷却时间 |

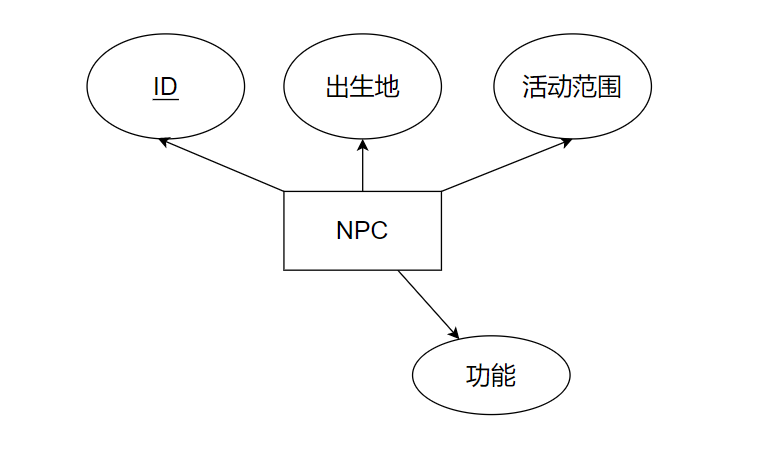
## 5.2数据库E-R图

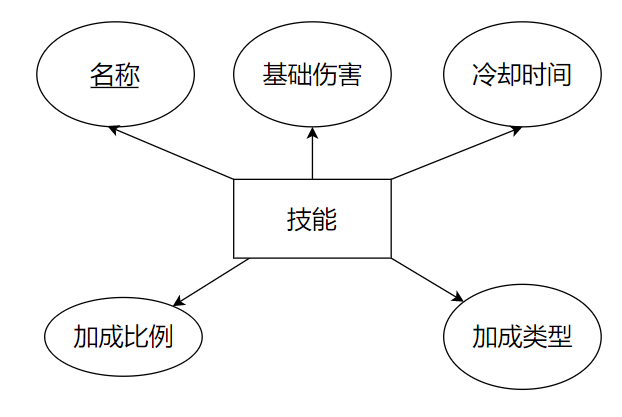


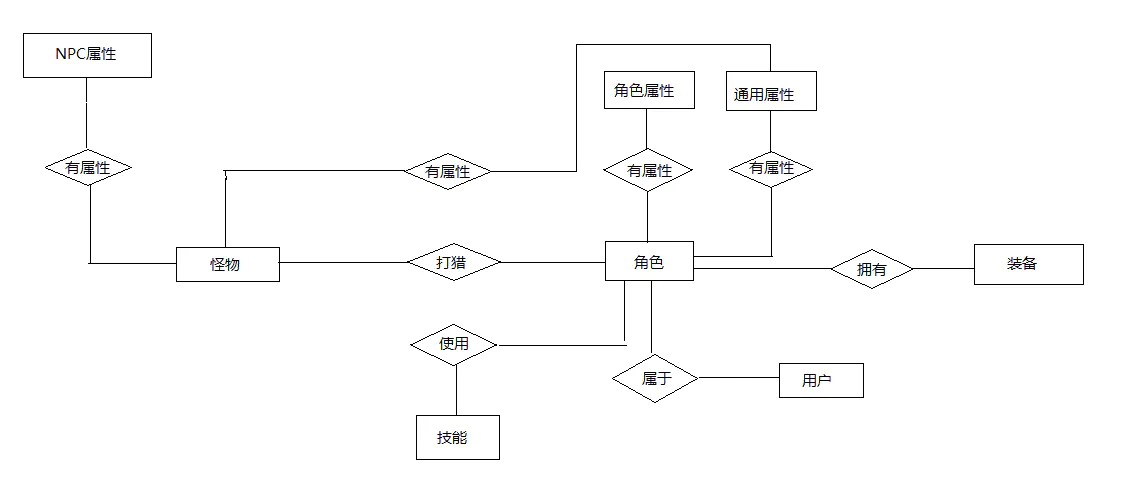






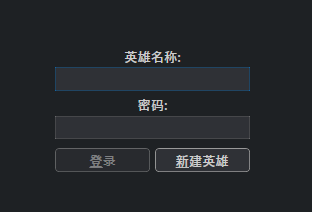






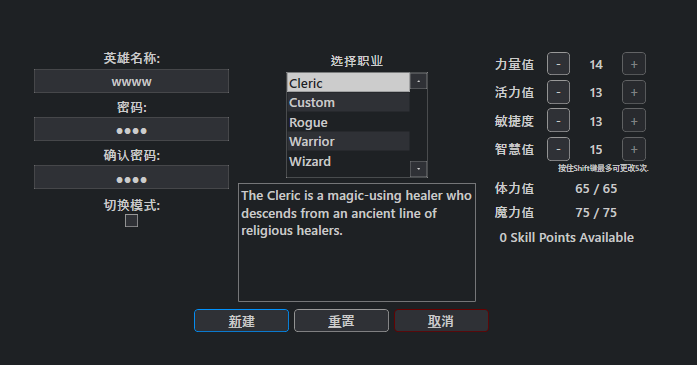
# 六 系统实现

## 6.1系统具体实现



第一次运行需要注册，需要用户名+密码，创建好的用户名和密码在sqllite数据库里，不可见。

点击注册进入注册主界面；

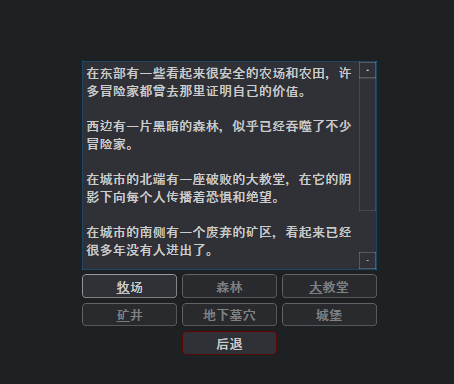


注册界面可以选择职业，并附带有相应的介绍，界面右侧有初始点数，可以随意进行分配，页面下方还有重置按钮。

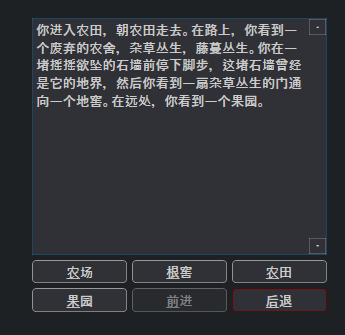
注册成功之后可以重新进行登陆。



进入游戏，最下方显示的依次是名字，体力值（生命值），魔力值，等级，经验和金币数量。

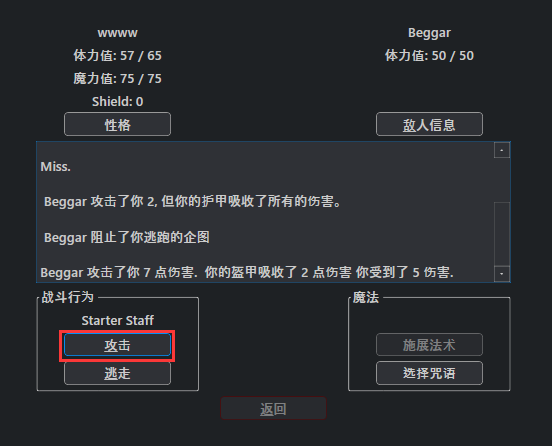


开始会随机安排故事。

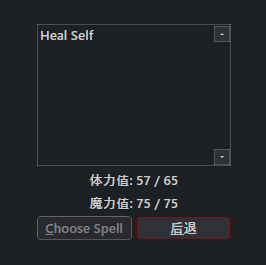


遇到敌人，开始进入战斗状态。



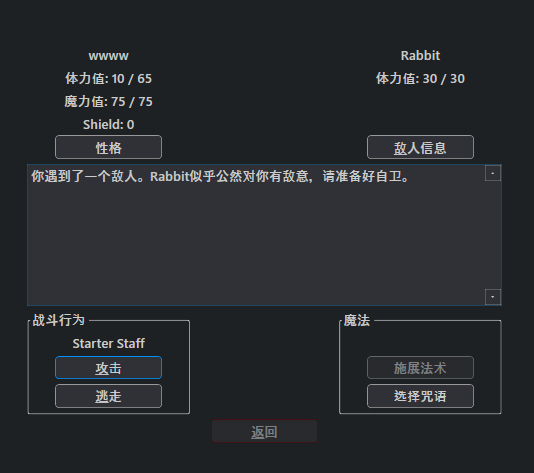












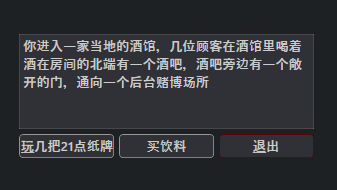


背包界面：



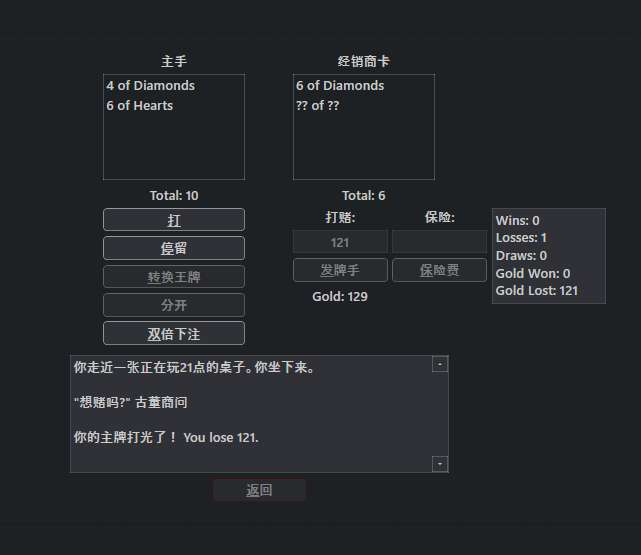


进入城市后可以选择进入酒馆：



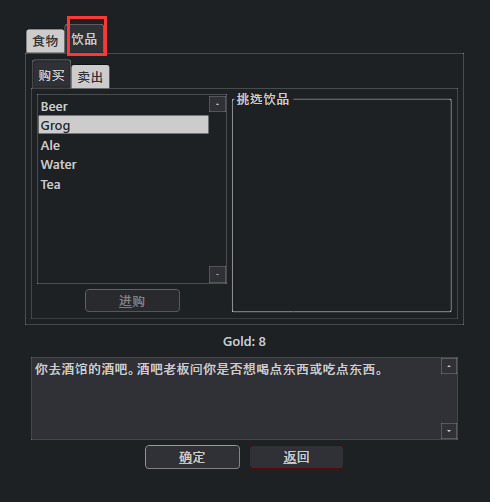
进入酒馆后可以进行游戏：



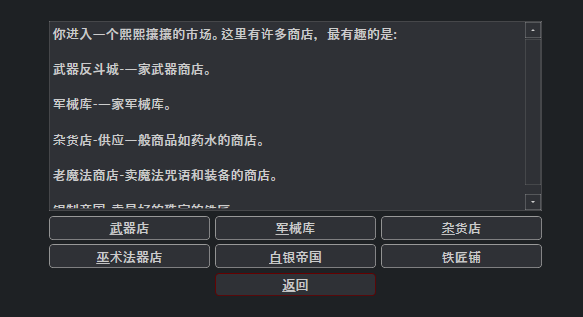


可以购买食物和饮品。





除了进入酒馆，还可以进入市场进行武器装备的买卖。有武器店，军械库，杂货店，巫术法器店，白银帝国（饰品店），铁匠铺。

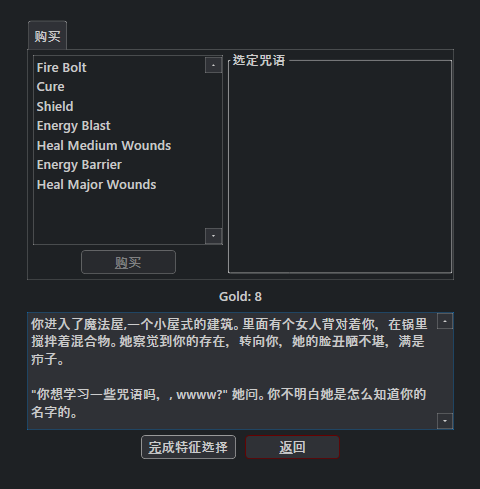








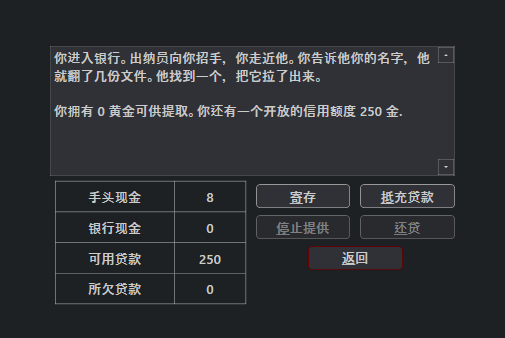


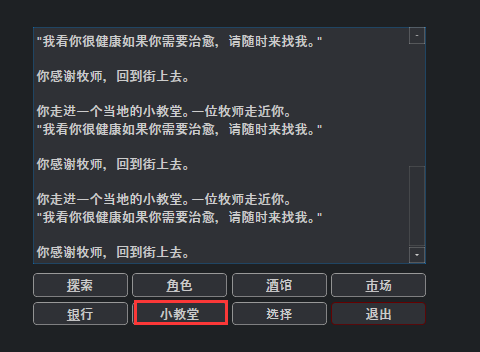


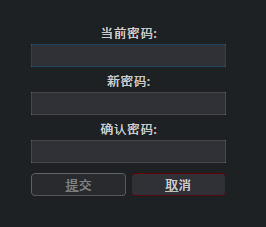


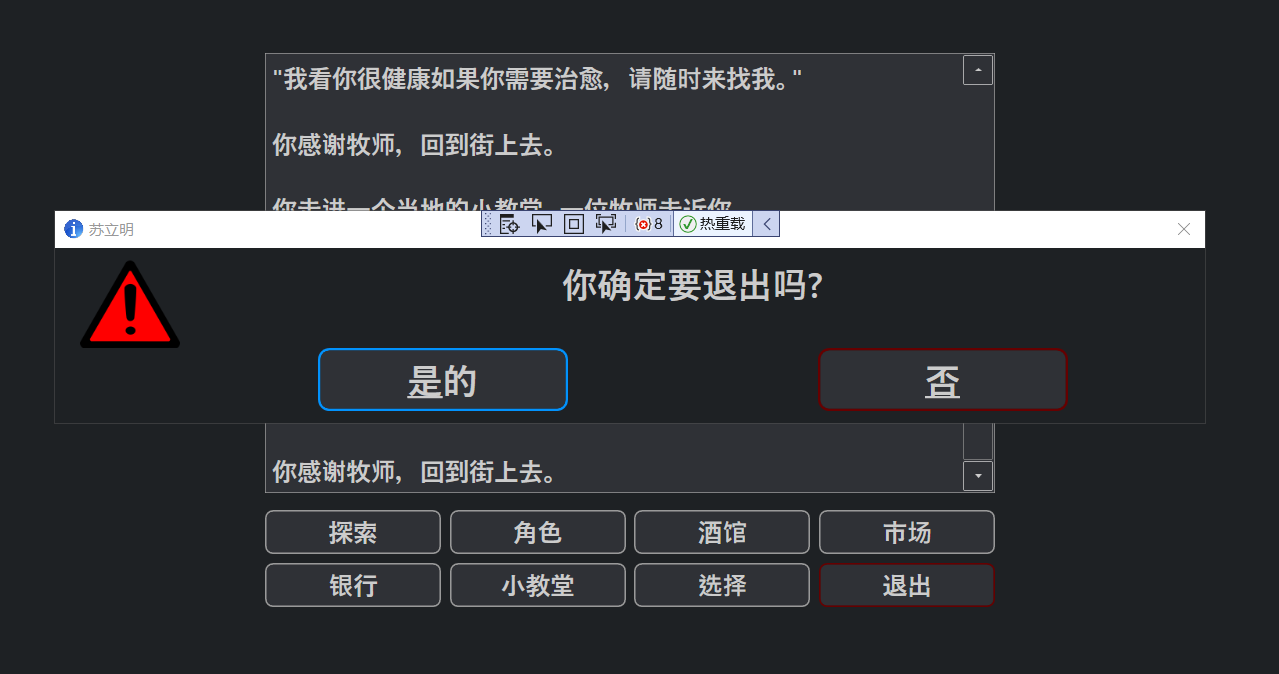


可以进入银行，进行寄存或者贷款。











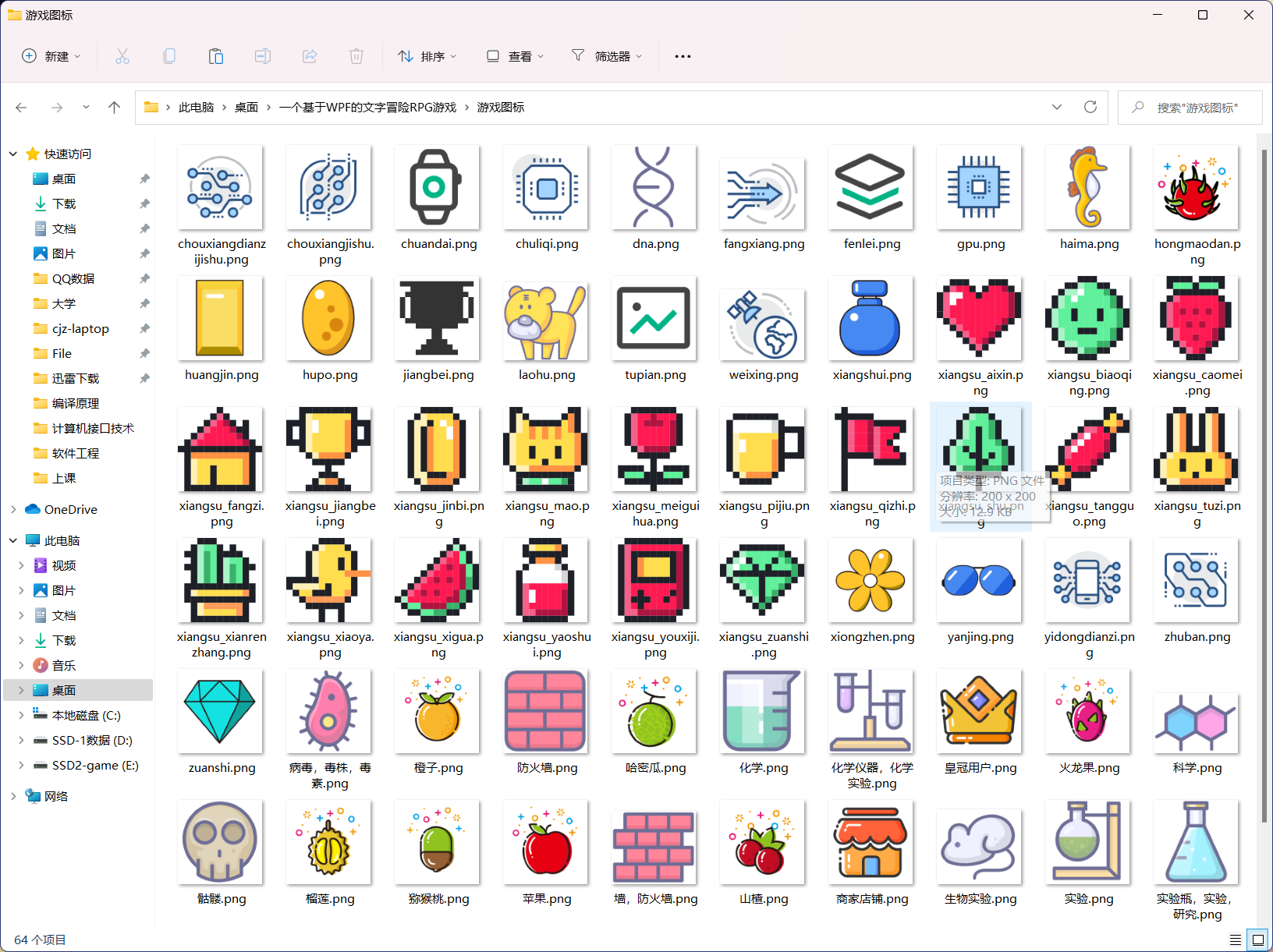
## 6.2项目改进

我们经过了修改，完善了背景颜色，使其可以进行选择：





我们还添加了一些图标：



# 六 心得体会

我们的小游戏主要通过模拟不同职业在不同情景下的动作来带给玩家新鲜感，软件中随机事件支持通过继承基类的方式添加，在游戏中，可以根据自己的想法设计随机事件的触发权重，造成的效果，以及触发时显示的文案，玩家也可以自定义剧本，只需要生成对应格式的文件即可，各类事件的文案也可以根据自己喜好随意增删修改，整体上功能较全。

通过完成大作业，我们将这个学期所学的C#有关知识运用起来，它能锻炼我们的实践能力，也是对我们的知识学习的一次考察。在选定题目后，我们就在网上找了许多相关的资料文档，开始设计时，我们先做了整体的分析，把要设计的表，表的属性，表间的关系全部弄清楚，然后分析系统需要的功能，用了SQLite数据库，并画出了整个系统的功能模块图和E-R图,使得这次课程设计的整体构架基本展现出来，使用VS2019搭建windows窗体，制作界面，并连接到数据库，完成整个系统的制作，我们了解了很多知识，增强了自身的实践能力。

于此同时，我们也从中发现很多的不足，首先，看到课题后的很长一段时间我一直没有认真仔细地思考过需要怎么设计好这个数据库，前台窗体界面应该怎样去设计，当准备开始动手的时候，前期工作造成开始的分工不合理，系统具体实现比较混乱，再次通过细致的分层，是逻辑清晰，是后期调用思路也比较简单，但一些地方的逻辑不是很严谨，系统界面不是很美观，过场展示仅限文字，但事实上过场增加有趣的图片显示是完全可以的，而且使用时提示不是特别及时，对于改进，我们会进一步设计有意思的随机事件，丰富各个事件的文案内容，从而尽可能避免重复的文案出现。

# 七 参考文献

1. 李佳,付强,丁宁. C#开发技术大全[M] . 北京:清华大学出版社,2009-2
2. 康 博.simon robinson,ollie cornes. c#高级编程

# 八 附录（核心源代码）

## SQLiteHelper.cs

using Extensions.DataTypeHelpers;

using Extensions.Enums;

using System;

using System.Data;

using System.Data.SQLite;

using System.Threading.Tasks;

using System.Windows;

namespace Extensions.DatabaseHelp

{

/// <summary>Provides an extension into SQLite commands.</summary>

public static class SQLiteHelper

{

/// <summary>This method fills a DataSet with data from a table.</summary>

/// <param name="con">Connection information</param>

/// <param name="sql">SQL query to be executed</param>

/// <returns>Returns DataSet with queried results</returns>

public static Task<DataSet> FillDataSet(string con, string sql) => FillDataSet(con, new SQLiteCommand { CommandText = sql });

/// <summary>This method fills a DataSet with data from a table.</summary>

/// <param name="con">Connection information</param>

/// <param name="cmd">SQLite command to be executed</param>

/// <returns>Returns DataSet with queried results</returns>

public static async Task<DataSet> FillDataSet(string con, SQLiteCommand cmd)

{

DataSet ds = new DataSet();

SQLiteConnection connection = new SQLiteConnection(con);

cmd.Connection = connection;

await Task.Run(() =>

{

try

{

SQLiteDataAdapter da = new SQLiteDataAdapter(cmd);

da.Fill(ds);

}

catch (SQLiteException ex)

{

Application.Current.Dispatcher.Invoke(() => new Notification(ex.Message, "Error Filling DataSet", NotificationButton.OK).ShowDialog());

}

finally

{

connection.Close();

}

}).ConfigureAwait(false);

return ds;

}

/// <summary>This method fills a DataSet with data from a table.</summary>

/// <param name="cmd">SQLite command to be executed</param>

/// <param name="con">Connection information</param>

/// <returns>Returns DataSet with queried results</returns>

[Obsolete("This method has its parameters backwards. Use FillDataSet(string con, SQLiteCommand cmd) instead.", true)]

public static async Task<DataSet> FillDataSet(SQLiteCommand cmd, string con)

{

DataSet ds = new DataSet();

SQLiteConnection connection = new SQLiteConnection(con);

cmd.Connection = connection;

await Task.Run(() =>

{

try

{

SQLiteDataAdapter da = new SQLiteDataAdapter(cmd);

da.Fill(ds);

}

catch (SQLiteException ex)

{

Application.Current.Dispatcher.Invoke(() => new Notification(ex.Message, "Error Filling DataSet", NotificationButton.OK).ShowDialog());

}

finally

{

connection.Close();

}

}).ConfigureAwait(false);

return ds;

}

/// <summary>Gets the next index from the SQLITE\_SEQUENCE table for a passed table's autoincrement value</summary>

/// <param name="connectionString">Connection string for the database</param>

/// <param name="tableName">Name of the table whose autoincrement value being requested</param>

/// <returns>Autoincrement value being requested</returns>

public static async Task<int> GetNextIndex(string connectionString, string tableName)

{

DataSet ds = await FillDataSet(connectionString, $"SELECT \* FROM SQLITE\_SEQUENCE WHERE [name] = '{tableName}'");

return ds.Tables[0].Rows.Count > 0 ? Int32Helper.Parse(ds.Tables[0].Rows[0]["seq"]) + 1 : 1;

}

/// <summary>Executes commands.</summary>

/// <param name="con">Connection information</param>

/// <param name="commands">Commands to be executed</param>

/// <returns>Returns true if command(s) executed successfully</returns>

public static async Task<bool> ExecuteCommand(string con, params SQLiteCommand[] commands)

{

bool success = false;

if (!string.IsNullOrWhiteSpace(con))

{

SQLiteConnection connection = new SQLiteConnection(con);

await Task.Run(() =>

{

try

{

connection.Open();

foreach (SQLiteCommand command in commands)

{

command.Connection = connection;

command.Prepare();

command.ExecuteNonQuery();

}

success = true;

}

catch (SQLiteException ex)

{

Application.Current.Dispatcher.Invoke(() =>

{

new Notification(ex.Message, "Error Executing Command", NotificationButton.OK)

.ShowDialog();

});

}

finally

{

connection.Close();

}

}).ConfigureAwait(false);

}

else

{

Application.Current.Dispatcher.Invoke(() =>

{

new Notification("Connection string cannot be empty!", "Cannot Connect To Database",

NotificationButton.OK).ShowDialog();

});

}

return success;

}

}

}

## Card.cs

using Sulimn.Classes.Enums;

using System;

namespace Sulimn.Classes.Card

{

/// <summary>Represents a playing card.</summary>

internal class Card : BaseINPC, IEquatable<Card>

{

private string \_name;

private CardSuit \_suit;

private int \_value;

private bool \_hidden;

#region Properties

/// <summary>The name of the card.</summary>

public string Name

{

get => \_name;

set

{

\_name = value;

NotifyPropertyChanged(nameof(Name));

}

}

/// <summary>The suit of the card.</summary>

public CardSuit Suit

{

get => \_suit;

set

{

\_suit = value;

NotifyPropertyChanged(nameof(Suit));

}

}

/// <summary>The value of the card.</summary>

public int Value

{

get => \_value;

set

{

\_value = value;

NotifyPropertyChanged(nameof(Value));

}

}

/// <summary>Should the Card be hidden from the player?</summary>

public bool Hidden

{

get => \_hidden;

set

{

\_hidden = value;

NotifyPropertyChanged(nameof(Hidden));

}

}

/// <summary>Returns the name and suit of the card.</summary>

public string CardToString => Hidden ? "?? of ??" : $"{Name} of {Suit}";

#endregion Properties

#region Override Operators

public static bool Equals(Card left, Card right)

{

if (left is null && right is null) return true;

if (left is null ^ right is null) return false;

return string.Equals(left.Name, right.Name, StringComparison.OrdinalIgnoreCase)

&& left.Suit == right.Suit

&& left.Value == right.Value && left.Hidden == right.Hidden;

}

public sealed override bool Equals(object obj) => Equals(this, obj as Card);

public bool Equals(Card other) => Equals(this, other);

public static bool operator ==(Card left, Card right) => Equals(left, right);

public static bool operator !=(Card left, Card right) => !Equals(left, right);

public sealed override int GetHashCode() => base.GetHashCode() ^ 17;

public sealed override string ToString() => CardToString;

#endregion Override Operators

#region Constructors

/// <summary>Initializes a default instance of Card.</summary>

internal Card()

{

}

/// <summary>Initializes an instance of Card by assigning Properties.</summary>

/// <param name="name">Name of Card</param>

/// <param name="suit">Suit of Card</param>

/// <param name="value">Value of Card</param>

/// <param name="hidden">Should the Card be hidden from the player?</param>

internal Card(string name, CardSuit suit, int value, bool hidden)

{

Name = name;

Suit = suit;

Value = value;

Hidden = hidden;

}

/// <summary>Replaces this instance of Card with another instance.</summary>

/// <param name="other">Instance to replace this instance</param>

internal Card(Card other) : this(other.Name, other.Suit, other.Value, other.Hidden)

{

}

/// <summary>Replaces this instance of Card with another instance.</summary>

/// <param name="other">Instance to replace this instance</param>

/// <param name="hidden">Should the Card be hidden from the player?</param>

internal Card(Card other, bool hidden) : this(other.Name, other.Suit, other.Value, hidden)

{

}

#endregion Constructors

}

}

## Hand.cs

using System.Collections.Generic;

using System.Linq;

namespace Sulimn.Classes.Card

{

/// <summary>Represents a hand of playing cards.</summary>

internal class Hand : BaseINPC

{

private List<Card> \_cardList = new List<Card>();

#region Modifyering Properties

/// <summary>List of <see cref="Card"/>s in the <see cref="Hand"/>.</summary>

public List<Card> CardList

{

get => \_cardList;

set

{

\_cardList = value;

UpdateProperties();

}

}

#endregion Modifyering Properties

#region Helper Properties

/// <summary>Actual value of <see cref="Card"/>s in the <see cref="Hand"/>.</summary>

/// <returns>Actual value</returns>

internal int ActualValue => \_cardList.Sum(card => card.Value);

/// <summary>Total value of <see cref="Card"/>s in <see cref="Hand"/>, excluding Hidden <see cref="Card"/>s.</summary>

/// <returns>Total value</returns>

internal int TotalValue => \_cardList.Where(card => !card.Hidden).Sum(card => card.Value);

/// <summary>Count of <see cref="Card"/>s in the <see cref="Hand"/>.</summary>

internal int Count => CardList.Count;

/// <summary>Current total value of the <see cref="Hand"/>, with preceding text.</summary>

public string Value => TotalValue > 0 ? $"Total: {TotalValue}" : "";

#endregion Helper Properties

#region Hand Management

/// <summary>Adds a <see cref="Card"/>s to the <see cref="Hand"/>.</summary>

/// <param name="newCard">Card to be added.</param>

internal void AddCard(Card newCard)

{

CardList.Add(newCard);

UpdateProperties();

}

/// <summary>Can the <see cref="Hand"/> be split?</summary>

/// <returns>True if <see cref="Hand"/> can be split</returns>

internal bool CanSplit() => CardList.Count == 2 && CardList[0].Name == CardList[1].Name && CardList[0].Value == CardList[1].Value;

/// <summary>Checks whether the <see cref="Hand"/> can be Doubled Down.</summary>

/// <returns>Returns true if <see cref="Hand"/> can be Doubled Down</returns>

internal bool CanDoubleDown() => CardList.Count == 2 && ActualValue >= 9 && ActualValue <= 11;

/// <summary>Clears the Hidden state of all <see cref="Card"/>s in the <see cref="Hand"/>.</summary>

internal void ClearHidden()

{

foreach (Card card in CardList)

card.Hidden = false;

UpdateProperties();

}

/// <summary>Converts an 11-valued Ace <see cref="Card"/> to be valued at 1.</summary>

internal void ConvertAce()

{

foreach (Card card in CardList)

if (card.Value == 11)

{

card.Value = 1;

break;

}

UpdateProperties();

}

/// <summary>Checks whether a <see cref="Hand"/> has an Ace valued at eleven in it.</summary>

/// <returns>Returns true if <see cref="Hand"/> has an Ace valued at eleven in it</returns>

internal bool HasAceEleven() => CardList.Any(card => card.Value == 11);

/// <summary>Checks whether a <see cref="Hand"/> has reached Blackjack.</summary>

/// <returns>Returns true if the <see cref="Hand"/>'s value is 21</returns>

internal bool HasBlackjack() => ActualValue == 21;

/// <summary>Checks whether the <see cref="Hand"/> has a Five Card Charlie.</summary>

/// <returns>Returns true if the <see cref="Hand"/> has a Five Card Charlie</returns>

internal bool HasFiveCardCharlie() => Count == 5 && (ActualValue < 21 || (HasAceEleven() && ActualValue <= 31));

/// <summary>Checks whether the current <see cref="Hand"/> has gone Bust.</summary>

/// <returns>Returns true if <see cref="Hand"/> has gone Bust</returns>

internal bool IsBust() => !HasAceEleven() && ActualValue > 21;

/// <summary>Updates the 3 important Properties of the Hand.</summary>

private void UpdateProperties() => NotifyPropertyChanged(nameof(ActualValue), nameof(CardList), nameof(Count), nameof(TotalValue), nameof(Value));

#endregion Hand Management

#region Constructors

/// <summary>Initializes a default instance of <see cref="Hand"/>.</summary>

internal Hand()

{

}

/// <summary>Initializes an instance of <see cref="Hand"/> by assigning the list of <see cref="Card"/>s.</summary>

/// <param name="cardList">List of <see cref="Card"/>s</param>

internal Hand(List<Card> cardList) => CardList = cardList;

#endregion Constructors

}

}

## Item.cs

using Extensions.DataTypeHelpers;

using Newtonsoft.Json;

using Sulimn.Classes.Entities;

using Sulimn.Classes.HeroParts;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Sulimn.Classes.Items

{

/// <summary>Represents an <see cref="Item"/> that an entity can interact with in the game.</summary>

public class Item : BaseINPC

{

private string name, description, texture;

private ItemType type;

private int damage, defense, weight, value, currentDurability, maximumDurability, strength, vitality, dexterity, wisdom, restoreHealth, restoreMagic, minimumLevel;

private bool cures, canSell, isSold;

private List<HeroClass> allowedClasses = new List<HeroClass>();

#region Modifying Properties

/// <summary>Name of the <see cref="Item"/>.</summary>

[JsonProperty(Order = 1)]

public string Name

{

get => name;

set { name = value; NotifyPropertyChanged(nameof(Name)); }

}

/// <summary>Description of the <see cref="Item"/>.</summary>

[JsonProperty(Order = 2)]

public string Description

{

get => description;

set { description = value; NotifyPropertyChanged(nameof(Description)); }

}

/// <summary>Type of the <see cref="Item"/>.</summary>

[JsonProperty(Order = 3)]

public ItemType Type

{

get => type;

set { type = value; NotifyPropertyChanged(nameof(Type), nameof(TypeToString)); }

}

/// <summary>Amount of damage the <see cref="Item"/> inflicts.</summary>

[JsonProperty(Order = 4)]

public int Damage

{

get => damage;

set { damage = value; NotifyPropertyChanged(nameof(Damage), nameof(DamageToString), nameof(DamageToStringWithText)); }

}

/// <summary>Amount of damage the <see cref="Item"/> can defend against.</summary>

[JsonProperty(Order = 5)]

public int Defense

{

get => defense;

set { defense = value; NotifyPropertyChanged(nameof(Defense), nameof(DefenseToString), nameof(DefenseToStringWithText)); }

}

/// <summary>Amount the <see cref="Item"/> weighs.</summary>

[JsonProperty(Order = 6)]

public int Weight

{

get => weight;

set { weight = value; NotifyPropertyChanged(nameof(Weight), nameof(WeightToString), nameof(WeightToStringWithText), nameof(Weight)); }

}

/// <summary>Amount the <see cref="Item"/> is worth.</summary>

[JsonProperty(Order = 7)]

public int Value

{

get => value;

set { this.value = value; NotifyPropertyChanged(nameof(Value), nameof(ValueToString), nameof(ValueToStringWithText), nameof(SellValue), nameof(SellValueToString), nameof(SellValueToStringWithText)); }

}

/// <summary>The current durability of an <see cref="Item"/>.</summary>

[JsonProperty(Order = 8)]

public int CurrentDurability

{

get => currentDurability;

set { currentDurability = value; NotifyPropertyChanged(nameof(Durability), nameof(CurrentDurability), nameof(CurrentDurabilityToString), nameof(DurabilityRatio), nameof(DurabilityString)); }

}

/// <summary>The maximum durability of an <see cref="Item"/>.</summary>

[JsonProperty(Order = 9)]

public int MaximumDurability

{

get => maximumDurability;

set { maximumDurability = value; NotifyPropertyChanged(nameof(Durability), nameof(MaximumDurability), nameof(MaximumDurabilityToString), nameof(DurabilityRatio), nameof(DurabilityString)); }

}

/// <summary>Amount of bonus Strength the <see cref="Item"/> grants.</summary>

[JsonProperty(Order = 10)]

public int Strength

{

get => strength;

set { strength = value; NotifyPropertyChanged(nameof(Strength), nameof(StrengthToString)); }

}

/// <summary>Amount of bonus Vitality the <see cref="Item"/> grants.</summary>

[JsonProperty(Order = 11)]

public int Vitality

{

get => vitality;

set { vitality = value; NotifyPropertyChanged(nameof(Vitality), nameof(VitalityToString)); }

}

/// <summary>Amount of bonus Dexterity the <see cref="Item"/> grants.</summary>

[JsonProperty(Order = 12)]

public int Dexterity

{

get => dexterity;

set { dexterity = value; NotifyPropertyChanged(nameof(Dexterity), nameof(DexterityToString)); }

}

/// <summary>Amount of bonus Wisdom the <see cref="Item"/> grants.</summary>

[JsonProperty(Order = 13)]

public int Wisdom

{

get => wisdom;

set { wisdom = value; NotifyPropertyChanged(nameof(Wisdom), nameof(WisdomToString)); }

}

/// <summary>Amount of Health this <see cref="Item"/> restores.</summary>

[JsonProperty(Order = 14)]

public int RestoreHealth

{

get => restoreHealth;

set { restoreHealth = value; NotifyPropertyChanged(nameof(RestoreHealth), nameof(RestoreHealthToString)); }

}

/// <summary>Amount of Magic this <see cref="Item"/> restores.</summary>

[JsonProperty(Order = 15)]

public int RestoreMagic

{

get => restoreMagic;

set { restoreMagic = value; NotifyPropertyChanged(nameof(RestoreMagic), nameof(RestoreMagicToString)); }

}

/// <summary>Does this <see cref="Item"/> cure?</summary>

[JsonProperty(Order = 16)]

public bool Cures

{

get => cures;

set { cures = value; NotifyPropertyChanged(nameof(Cures), nameof(CuresToString)); }

}

/// <summary>The minimum level a <see cref="Hero"/> is required to be to use the <see cref="Item"/>.</summary>

[JsonProperty(Order = 17)]

public int MinimumLevel

{

get => minimumLevel;

set { minimumLevel = value; NotifyPropertyChanged(nameof(MinimumLevel), nameof(MinimumLevelToString)); }

}

/// <summary>Can the <see cref="Item"/> be sold to a shop?</summary>

[JsonProperty(Order = 18)]

public bool CanSell

{

get => canSell;

set { canSell = value; NotifyPropertyChanged(nameof(CanSell), nameof(CanSellToString)); }

}

/// <summary>Is the <see cref="Item"/> sold in a shop?</summary>

[JsonProperty(Order = 19)]

public bool IsSold

{

get => isSold;

set { isSold = value; NotifyPropertyChanged(nameof(IsSold)); }

}

/// <summary><see cref="HeroClass"/>es permitted to use this <see cref="Item"/>.</summary>

[JsonIgnore]

public List<HeroClass> AllowedClasses

{

get => allowedClasses;

set { allowedClasses = value; NotifyPropertyChanged(nameof(AllowedClasses), nameof(AllowedClassesToString)); }

}

/// <summary><see cref="HeroClass"/>es allowed to use the <see cref="Item"/>, set up to import from JSON.</summary>

[JsonProperty(Order = 20)]

public string AllowedClassesJson

{

get => AllowedClasses?.Count > 0 ? string.Join(",", AllowedClasses) : "";

set

{

if (!string.IsNullOrWhiteSpace(value))

{

AllowedClasses = new List<HeroClass>();

AllowedClasses.AddRange(from string itm in value.Split(',') select GameState.AllClasses.Find(cls => cls.Name == itm.Trim()));

NotifyPropertyChanged(nameof(AllowedClasses), nameof(AllowedClassesToString));

}

}

}

/// <summary>Location of the Texture to be used for this <see cref="Item"/>.</summary>

[JsonProperty(Order = 21)]

public string Texture

{

get => texture;

set { texture = value; NotifyPropertyChanged(nameof(Texture)); }

}

#endregion Modifying Properties

#region Helper Properties

/// <summary>Damage the <see cref="Item"/> inflicts, formatted.</summary>

[JsonIgnore]

public string DamageToString => Damage.ToString("N0");

/// <summary>Damage the <see cref="Item"/> inflicts, formatted, with preceding text.</summary>

[JsonIgnore]

public string DamageToStringWithText => Damage > 0 ? $"Damage: {DamageToString}" : "";

/// <summary>Amount of damage the <see cref="Item"/> can defend against, formatted.</summary>

[JsonIgnore]

public string DefenseToString => Defense.ToString("N0");

/// <summary>Amount of damage the <see cref="Item"/> can defend against, formatted, with preceding text.</summary>

[JsonIgnore]

public string DefenseToStringWithText => Defense > 0 ? $"Defense: {DefenseToString}" : "";

/// <summary>Returns text relating to the amount of Health restored by the <see cref="Consumable"/>.</summary>

[JsonIgnore]

public string RestoreHealthToString => RestoreHealth > 0 ? $"Restores {RestoreHealth:N0} Health." : "";

/// <summary>Returns text relating to the amount of Magic restored by the <see cref="Consumable"/>.</summary>

[JsonIgnore]

public string RestoreMagicToString => RestoreMagic > 0 ? $"Restores {RestoreMagic:N0} Magic." : "";

/// <summary>Minimum level the <see cref="Hero"/> need to be to use the <see cref="Item"/>, with preceding text.</summary>

[JsonIgnore]

public string MinimumLevelToString => $"Minimum Level: {MinimumLevel}";

/// <summary>Returns text regarding if the <see cref="Consumable"/> can heal the user.</summary>

[JsonIgnore]

public string CuresToString => Cures ? $"Cures ailments." : "";

/// <summary>Returns text regarding all effects this <see cref="Consumable"/> will induce.</summary>

[JsonIgnore]

public string EffectsToString

{

get

{

string effects = "";

if (RestoreHealth > 0)

{

effects += RestoreHealthToString;

}

if (RestoreMagic > 0)

{

if (effects.Length > 0)

effects += "\n";

effects += RestoreMagicToString;

}

if (Cures)

{

if (effects.Length > 0)

effects += "\n";

effects += CuresToString;

}

return effects;

}

}

/// <summary>The current durability of an <see cref="Item"/>, with thousands separators.</summary>

[JsonIgnore]

public string CurrentDurabilityToString => CurrentDurability.ToString("N0");

/// <summary>The maximum durability of an <see cref="Item"/>, with thousands separators.</summary>

[JsonIgnore]

public string MaximumDurabilityToString => MaximumDurability.ToString("N0");

/// <summary>The durability of an <see cref="Item"/>, formatted.</summary>

[JsonIgnore]

public string Durability => $"{CurrentDurabilityToString} / {MaximumDurabilityToString}";

/// <summary>The durability of an <see cref="Item"/>, formatted.</summary>

[JsonIgnore]

public string DurabilityString => $"Durability: {Durability}";

/// <summary>The durability ratio of an <see cref="Item"/>.</summary>

[JsonIgnore]

public decimal DurabilityRatio => MaximumDurability > 0 ? CurrentDurability \* 1m / MaximumDurability : 0;

/// <summary>The weight of the <see cref="Item"/> with thousands separators.</summary>

[JsonIgnore]

public string WeightToString => Weight.ToString("N0");

/// <summary>The weight of the <see cref="Item"/> with thousands separators and preceding text.</summary>

[JsonIgnore]

public string WeightToStringWithText => $"Weight: {WeightToString}";

/// <summary>The value of the <see cref="Item"/> with thousands separators.</summary>

[JsonIgnore]

public string ValueToString => Value.ToString("N0");

/// <summary>The value of the <see cref="Item"/> with thousands separators and preceding text.</summary>

[JsonIgnore]

public string ValueToStringWithText => !string.IsNullOrWhiteSpace(Name) ? $"Value: {ValueToString}" : "";

/// <summary>The value of the Item.</summary>

[JsonIgnore]

public int SellValue => MaximumDurability > 0 ? Value / 2 \* CurrentDurability / MaximumDurability : 0;

/// <summary>The sell value of the <see cref="Item"/> with thousands separators.</summary>

[JsonIgnore]

public string SellValueToString => SellValue.ToString("N0");

/// <summary>The sell value of the <see cref="Item"/> with thousands separators with preceding text.</summary>

[JsonIgnore]

public string SellValueToStringWithText => !string.IsNullOrWhiteSpace(Name) ? $"Sell Value: {SellValueToString}" : "";

/// <summary>The amount of gold it would cost to repair this <see cref="Item"/>.</summary>

[JsonIgnore]

public int RepairCost => Int32Helper.Parse((1 - DurabilityRatio) \* Value);

/// <summary>The amount of gold it would cost to repair this <see cref="Item"/>,formatted.</summary>

[JsonIgnore]

public string RepairCostToString => RepairCost.ToString("N0");

/// <summary>The amount of gold it would cost to repair this <see cref="Item"/>, formatted with preceding text.</summary>

[JsonIgnore]

public string RepairCostToStringWithText => RepairCost > 0 ? $"Repair Cost: {RepairCostToString}" : "";

/// <summary>Returns text relating to the sellability of the <see cref="Item"/>.</summary>

[JsonIgnore]

public string CanSellToString => !string.IsNullOrWhiteSpace(Name) ? (CanSell ? "Sellable" : "Not Sellable") : "";

/// <summary>Returns the Strength and preceding text.</summary>

[JsonIgnore]

public string StrengthToString => Strength > 0 ? $"Strength: {Strength}" : "";

/// <summary>Returns the Vitality and preceding text.</summary>

[JsonIgnore]

public string VitalityToString => Vitality > 0 ? $"Vitality: {Vitality}" : "";

/// <summary>Returns the Dexterity and preceding text.</summary>

[JsonIgnore]

public string DexterityToString => Dexterity > 0 ? $"Dexterity: {Dexterity}" : "";

/// <summary>Returns the Wisdom and preceding text.</summary>

[JsonIgnore]

public string WisdomToString => Wisdom > 0 ? $"Wisdom: {Wisdom}" : "";

/// <summary>Returns all bonuses in string format.</summary>

[JsonIgnore]

public string BonusToString

{

get

{

string[] bonuses =

{

StrengthToString, VitalityToString,

DexterityToString, WisdomToString

};

return string.Join("\n", bonuses.Where(bonus => bonus.Length > 0));

}

}

/// <summary>Displays the Type of the <see cref="Item"/>, formatted.</summary>

[JsonIgnore]

public string TypeToString

{

get

{

switch (Type)

{

case ItemType.MeleeWeapon:

return "Melee Weapon";

case ItemType.RangedWeapon:

return "Ranged Weapon";

case ItemType.HeadArmor:

return "Head Armor";

case ItemType.BodyArmor:

return "Body Armor";

case ItemType.HandArmor:

return "Hand Armor";

case ItemType.LegArmor:

return "Leg Armor";

case ItemType.FeetArmor:

return "Feet Armor";

default:

return Type.ToString();

}

}

}

/// <summary>Text to be displayed when hovering over an <see cref="Item"/>.</summary>

[JsonIgnore]

public string TooltipText => this != new Item()

? Name + "\n"

+ Description + "\n"

+ TypeToString + "\n"

+ ((Damage > 0) ? DamageToStringWithText + "\n" : "")

+ ((Defense > 0) ? DefenseToStringWithText + "\n" : "")

+ ((Weight > 0) ? WeightToStringWithText + "\n" : "")

+ ValueToStringWithText + "\n"

+ DurabilityString + "\n"

+ (EffectsToString.Length > 0 ? EffectsToString + "\n" : "")

+ (MinimumLevel > 1 ? MinimumLevelToString + "\n" : "")

+ (AllowedClasses.Count > 0 ? AllowedClassesToString + "\n" : "")

+ BonusToString

: "";

/// <summary><see cref="HeroClass"/>es allowed to use the <see cref="Item"/>, formatted</summary>

[JsonIgnore]

public string AllowedClassesToString => String.Join(",", AllowedClasses);

#endregion Helper Properties

#region Override Operators

public static bool Equals(Item left, Item right)

{

if (left is null && right is null) return true;

if (left is null ^ right is null) return false;

return string.Equals(left.Name, right.Name, StringComparison.OrdinalIgnoreCase)

&& string.Equals(left.Description, right.Description, StringComparison.OrdinalIgnoreCase)

&& left.Type == right.Type

&& left.Damage == right.Damage

&& left.Defense == right.Defense

&& left.Weight == right.Weight

&& left.Value == right.Value

&& left.CurrentDurability == right.CurrentDurability

&& left.MaximumDurability == right.MaximumDurability

&& left.Strength == right.Strength

&& left.Vitality == right.Vitality

&& left.Dexterity == right.Dexterity

&& left.Wisdom == right.Wisdom

&& left.RestoreHealth == right.RestoreHealth

&& left.RestoreMagic == right.RestoreMagic

&& left.Cures == right.Cures

&& left.MinimumLevel == right.MinimumLevel

&& left.CanSell == right.CanSell

&& left.IsSold == right.IsSold

&& !left.AllowedClasses.Except(right.AllowedClasses).Any()

&& !right.AllowedClasses.Except(left.AllowedClasses).Any();

}

public override bool Equals(object obj) => Equals(this, obj as Item);

public bool Equals(Item otherItem) => Equals(this, otherItem);

public static bool operator ==(Item left, Item right) => Equals(left, right);

public static bool operator !=(Item left, Item right) => !Equals(left, right);

public override int GetHashCode() => base.GetHashCode() ^ 17;

public override string ToString() => Name;

#endregion Override Operators

#region Constructors

/// <summary>Initializes a default instance of <see cref="Item"/>.</summary>

public Item()

{

}

/// <summary>Initializes an instance of <see cref="Item"/> by assigning values to Properties.</summary>

/// <param name="name">Name of the <see cref="Item"/></param>

/// <param name="description">Description of the <see cref="Item"/></param>

/// <param name="itemType">Type of the <see cref="Item"/></param>

/// <param name="damage">Amount of damage the <see cref="Item"/> inflicts</param>

/// <param name="defense">Amount of damage the <see cref="Item"/> can defend against</param>

/// <param name="weight">Amount the <see cref="Item"/> weighs</param>

/// <param name="value">Amount the <see cref="Item"/> is worth</param>

/// <param name="currentDurability">The current durability of an <see cref="Item"/></param>

/// <param name="maximumDurability">The maximum durability of an <see cref="Item"/></param>

/// <param name="strength">Amount of bonus Strength the <see cref="Item"/> grants</param>

/// <param name="vitality">Amount of bonus Vitality the <see cref="Item"/> grants</param>

/// <param name="dexterity">Amount of bonus Dexterity the <see cref="Item"/> grants</param>

/// <param name="wisdom">Amount of bonus Wisdom the <see cref="Item"/> grants</param>

/// <param name="restoreHealth">Amount of health this <see cref="Item"/> restores</param>

/// <param name="restoreMagic">Amount of magic this <see cref="Item"/> restores</param>

/// <param name="cures">Does this <see cref="Item"/> cure?</param>

/// <param name="minimumLevel">The minimum level a <see cref="Hero"/> is required to be to use the <see cref="Item"/></param>

/// <param name="canSell">Can the <see cref="Item"/> be sold to a shop?</param>

/// <param name="isSold">Is the <see cref="Item"/> sold in a shop?</param>

/// <param name="allowedClasses"><see cref="HeroClass"/>es permitted to use this <see cref="Item"/></param>

/// <param name="texture">Location of the Texture to be used for this <see cref="Item"/></param>

internal Item(string name, string description, ItemType itemType, int damage, int defense, int weight, int value, int currentDurability, int maximumDurability, int strength, int vitality, int dexterity, int wisdom, int restoreHealth, int restoreMagic, bool cures, int minimumLevel, bool canSell, bool isSold, List<HeroClass> allowedClasses, string texture)

{

Name = name;

Description = description;

Type = itemType;

Damage = damage;

Defense = defense;

Weight = weight;

Value = value;

CurrentDurability = currentDurability;

MaximumDurability = maximumDurability;

Strength = strength;

Vitality = vitality;

Dexterity = dexterity;

Wisdom = wisdom;

RestoreHealth = restoreHealth;

RestoreMagic = restoreMagic;

Cures = cures;

MinimumLevel = minimumLevel;

CanSell = canSell;

IsSold = isSold;

AllowedClasses = new List<HeroClass>(allowedClasses);

Texture = texture;

}

/// <summary>Replaces this instance of <see cref="Item"/> with another instance.</summary>

/// <param name="other">Instance of <see cref="Item"/> to replace this instance</param>

public Item(Item other) : this(other.Name, other.Description, other.Type, other.Damage, other.Defense, other.Weight, other.Value, other.CurrentDurability, other.MaximumDurability, other.Strength, other.Vitality, other.Dexterity, other.Wisdom, other.RestoreHealth, other.RestoreMagic, other.Cures, other.MinimumLevel, other.CanSell, other.IsSold, new List<HeroClass>(other.AllowedClasses), other.Texture)

{

}

#endregion Constructors

}

}