Super Auto Pets 程序设计文档

一、 项目介绍

1.1 引言

近几年来，电子游戏市场日益壮大，各类游戏层出不穷。而在这么多游戏中，自走棋无疑是一类兼具热度与娱乐性的热门游戏类型。具体来说，自走棋是战略游戏的子类型，其特点是类似于国际象棋的元素，玩家在准备阶段将角色放在网格状的战场上，然后在玩家无法直接操控的情况下与对方的角色战斗。在笔者看来，自走棋之所以能够这么流行，无论是 TX 旗下《银勺勺之战》的爆火，还是依靠《旅店战棋》重焕第二春的《炉×传说》，抑或是将塔防和自走棋做尝试性结合的《昨日圆车》，它们都凸显了自走棋的三个个重要特征：简单到奶奶都会的操作流程，复杂的阵容搭配和烧脑的策略布置以及极具随机性的战斗流程和强烈的视觉刺激。

简单的操作流程可以确保来自不同背景的每一位玩家都能在第一时间理解并上手这款游戏，换言之，通过降低的门槛来吸引更大的用户群体；当然另一方面也可以降低玩家的游玩成本，或者说，平衡不同玩家之间的实力。由于具体的战斗过程由电脑随机模拟完成，玩家不需要像 MOBA、RTS 游戏一样需要极高的操作技巧，手部乘区的效用大大削减。自走棋这种傻瓜式的设计理念，自然获得了只追求娱乐性而不是竞技性的玩家的青睐。

其实第二个特征的重点在于确保游戏只是操作起来简单，而不是内容也十分“简单”。玩过自走棋的玩家都知道，复杂的个体机制和数值，以及相应的团体连锁效应，对整个战局的影响巨大。这其实做到了玩家分化：躺平型玩家自然可以选择随便摆放阵容追求娱乐，策略型玩家则可以通过研究数值差异和阵容搭配来达到类实时竞技的效果。

最后一点因素笔者认为才是自走棋的精髓所在：玩游戏的第一要素是追求娱乐。而这种电脑模拟的随机性，其实达成了类似于赌博的爽感，再搭配上极具张力和冲击力的画面效果，自然可以大大促进……（一堆笔者不知道的化学物质）的释放，从而达成使玩家沉迷的最终目标。

1.2 初具雏形

写了这么多，笔者意图在于证明下面这个事实：**为什么我们小组要选择开发自走棋这一游戏类型**。

首先是降低程序编写难度，因为自走棋所需要接受玩家的外部输入非常少。而在所有外部输入之中，最重要其实只有两个：**购买和出售**。其他最多只是基于这两个最

基本操作延伸而出的附加操作。而当玩家完成了购买出售操作之后，剩下的全是程序内部的逻辑运行和运算，最终**自主执行战斗流程**。

其次是丰富的人物角色和技能体系，这大大方便了在编写过程中，我们小组成员可以随时在其中加入自己的创意，建立多个不同的独创角色；也易于通过不断增加角色，大大加强游戏的复杂度和可玩性，降低游戏的润色成本；同时最重要的一点是，方便做彩蛋——以我们小组的最终成果为例，我们设计了“**高雅人士**”这一互联网火爆形象，还有**铁牛**这一“**高大上**”的动物形象；并在食物中塞入了"**The cake is a lie**"这一游戏名梗。(具体出处是 2007 年电子游戏《传送门》)

最后的画面特效其实算是一项挑战任务：如何通过流畅且丰富的战斗动画来吸引教师和同学的眼球。虽然最终可能由于各种因素的影响，动画效果差强人意，但也充分体现了我们小组的实力（自豪脸）。

1.3 灵感来源

该项目的灵感来源或者说主要借鉴对象是一款叫做 **Super Auto Pets** 的游戏，它是一款由 Team Wood Games 开发并发行的自动战斗视频游戏，玩家选择拥有特殊能力的“宠物”来与其他玩家战斗。下面是一些简单的介绍：

游戏由多个回合组成，分为两个阶段：**准备阶段**和**战斗阶段**。

在准备阶段，玩家通过购买新宠物、喂养食物、提升等级以及将宠物位置从左向右切换来管理队伍，进入战斗阶段。玩家在每个准备阶段开始时拥有 10 金币，用于购买宠物、食物和刷新商店。玩家可以通过出售宠物或使用拥有金币能力的宠物技能来获得金币。玩家可以在这个阶段花费任意多的时间。

战斗阶段将在玩家完成准备阶段后开始。在此阶段，宠物之间自动对战，玩家无法控制。玩家队伍最右侧位置的宠物与对方队伍最左侧位置的宠物战斗。当宠物昏倒时，队伍最右侧位置的宠物会替代昏倒的宠物。此过程反复进行，直到一方或双方队伍的宠物都用尽，此时回合决定胜负或平局。如果选手获胜，将获得奖杯。如果输了，他们会失去一点生命值（奖杯数量保持不变）。如果平局，他们的生命值和奖杯保持不变。第三回合，如果玩家在前两回合内失去了一点生命值，则可恢复一点生命值。当玩家获得 10 个奖杯时，即为游戏获胜。

每两回合，玩家会晋升一个等级 ，获得更多宠物和食物，通常比之前更好。玩家从第一阶段开始，逐步晋升到第六阶段，之后不再有等级晋级。

玩家将与其他玩家的队伍或 AI 生成的队伍对战，如果该回合没有玩家。所有战斗均为异步进行。

而我们小组的 Super Auto Pets 保留了大部分原版的内容，但在这一基础上做了部分删改，详细内容请见下一部分。

当然，我们小组选择以这款游戏为原型蓝图，不仅仅是因为相较于市面上的其他自走棋游戏，这款游戏在游戏流程、角色设计、道具效果上做到了极致的精简，在保留游戏核心特征的同时做到了极强的娱乐性，还因为它有着一个丰富且完善的 wiki 网站，我们该项目的许多图片素材都是从上面获得的，这无疑大大降低了我们的美术开发成本，在此我们做出感谢声明

1.4 最终设计

由于我们小组不是专业的项目开发团队，再考虑到程序设计实训的大项目的时间限制，我们无法在这短暂的时间内完成商业级别的自走棋项目开发。于是在经过慎重考虑过后，我们小组决定保留支撑游戏核心体验的关键模块，忽略那些可以进一步增加游戏体验，但却不是必须的附加模块。保留的模块具体如下：

第一是**宠物系统**，在这个系统中，我们保留了**属性、等级、技能**这三个部分。其中属性只选择保留**血量**和**攻击**这两个最基本的属性；而等级相比于原作复杂的经验升级体系和随回合数增加解锁等级上限的功能，我们做出了简化——**两个相同的宠物便可以进行合并升级**，宠物等级的上限恒定为三级；而在技能这个最为复杂的功能上，我们小组简化了原作过于复杂的技能发动时机和发动条件，转变**在几个特定的时间点进行技能发动检验**（战斗开始前、宠物死亡时、战斗结束后），从而降低了实现难度。

第二是**战斗系统**，和原作一样，我们基本实现了回合制自动对战功能，包括两种模式——**单步执行和自动对战**。玩家既可以通过单击单步执行来观察每步作战的过程和结果，也可以选择自动作战来欣赏完整而连贯的整体作战过程。如果说还有可以完善和改进的地方，那就是相比与原作炫酷的技能发动特效和宠物对战时的特殊动画，我们只保留了简单的宠物平移碰撞，但也基本实现了战斗过程的可视化。

第三是**商店与食物系统**，这也是该游戏的核心功能，也是我们小组实现最完整的部分。在尊重原作的基础上，我们充分实现了下述的几个功能：**宠物的购买和出售**功能、**食物购买**并及时将效果运用在宠物上的功能、商店的**刷新功能**、特定宠物或食物的**冻结功能**、相同宠物的**升级功能**以及宠物队列顺序的**调换功能**。

第四是**图鉴系统**，在这里，玩家可以预览游戏中存在的所有宠物和食物。并且玩家可以自由从主界面和战斗页面中访问图鉴系统，为玩家提供了相当的灵活性。

第五是**战绩系统**，在这里，玩家可以查看自己的历史游玩记录。每次游玩都会详细地记录下游戏结束时玩家的奖杯数，并根据这个来进行战绩的排名，让玩家清楚的知道自己的最佳战绩。

当然，说完了我们小组保留下来的模块，笔者也提一下被我们小组精简的模块，而这恰恰也可以是我们该项目之后**可以优化和更新实现的部分**：

宠物属性扩展，更为细腻生动的动画模块，技能发动顺序逻辑判断、更为完善的

战局信息统计模块，游戏结束结算界面，联机对战、排行榜等复杂社交功能……

不可否认的是，通过我们小组所保留的模块，我们已经充分实现了一款基础自走棋的所有功能，已经达到了完成大项目所需的要求条件。

二、 整体框架介绍

为符合大项目的语言开发要求，我们小组按要求将 **C++**作为核心开发语言，同时选用了 **Qt 框架**进行 GUI 的开发。整个程序的框架可以分为三个模块，分别是 **engine 模块、model 模块和 ui 模块**，下面笔者将分模块进行详细介绍。

2.1 engine 模块：游戏的大脑

该模块负责游戏的核心驱动逻辑，是实现复杂规则的地方。由 BattleEngine.cpp 和 BattleEngine.hpp 这两个文件组成，它们的主要职责是封装并执行两个玩家队伍之间的自动战斗。设计该模块的原因如下所述：

第一是为了解决**封装复杂性**。众所周知，战斗是游戏中最复杂的环节，涉及到攻击、受伤、技能触发等一系列连锁反应。将所有这些逻辑封装在 BattleEngine 这一文件中，就可以使其他模块（如 BattleView）的实现变得非常简单。BattleView 只需向 BattleEngine 请求“下一步发生了什么”，然后将其播放出来即可，而无需关心战斗的具体裁决过程。

第二是方便**可测试性**。如前所述，设置一个单独的战斗引擎，从而可以实现独立的、自动化的单元测试。这样便于构造任意的对战阵容，从而方便战斗测试，寻找并修复可能潜在的错误，以确保所有技能交互的正确性。上述这一点这对于 GUI 应用来说是至关重要的。

第三是为了**可重放与可观察性**。getNextStep()接口的设计，使得战斗过程可以被一步步地分解。这不仅便于 BattleView 制作动画，也为未来可能的功能（如战斗回放）打下了基础。

2.2 model 模块：游戏的基石

该模块定义了构成游戏世界的所有实体，它们是纯粹的数据和行为的载体。具体体现在下述五个不同的方面（其中将 Pet 和 Food 合并介绍）：

2.2.1 Pet & Food

这两个实体的功能是将游戏中的核心概念“宠物”和“食物”抽象为类，从而方便我，嗯清晰地组织宠物的属性（攻击力、生命值）和行为逻辑（升级、被击败），并将食物的效果直接作用在宠物的属性上。

2.2.2 Player

设计这一部分的原因是为了将所有与单个玩家相关的数据（生命、金币、回合数、宠物队伍）聚合到一个 Player 类中，从而简化状态管理，方便后续战斗界面的编写。这样做到话无论是传递玩家数据给战斗引擎，还是进行游戏状态加载，我们都只需要处理一个 Player 对象就可以了，而不要面对一堆零散的变量。

2.2.3 Shop

由于商店本身具有复杂的状态和行为（例如，商品池、刷新机制、冻结状态），如果将这些逻辑分散在 Player 类或 ShopView 类中，会造成职责不清和代码混乱。所以我们单独创建了一个 Shop 类，让它专门负责管理商品和交易逻辑，使得 Player 只需与之交互（buy, roll），而 ShopView 只需展示 Shop 的当前状态，从而使得三者权责分明，互不干扰。

2.2.4 Skill

这一部分可以说是整个模型设计的关键之处，如果用专业术语来说，那就是体现了我们小组“组合优于继承”和 “策略模式”的思想——具体体现在以下两个方面：

首先是**避免了类爆炸**。如果我们为每一种有特殊技能的宠物都创建一个子类（如 AntPet、FishPet），那么随着宠物数量的增加，类的数量会急剧膨胀，难以管理，大大加大编程难度。

其次是增强了**灵活性和可扩展性**：我们小组将“技能”抽象成一个独立的 Skill 类，这样宠物类只需持有一个 Skill 对象的列表。当需要创造一个新技能时，我们只需创建一个新的 Skill 实例或子类，然后将其“附加”给任意宠物即可，而无需修改宠物类本身。这使得添加新宠物、新食物变得异常简单。除此之外，Skill 类定义了技能的触发时机（如“攻击前”、“被击败后”）和具体效果，战斗引擎只需要在相应时机检查并触发这些技能即可。

2.3 ui 模块：玩家的窗口

该模块的主要功能是负责用户界面的实现，它由 StartView、ShopView、BattleView 等文件构成。设计该模块的原因是为了将游戏的不同阶段或屏幕拆分为独立的视图类，从而方便管理复杂的 UI。下面挑选三个主要的特点进行介绍：

第一点是实现了**单一职责**，即每个视图类只负责一个特定的界面。举例来说，ShopView 只关心商店和队伍管理，BattleView 只关心战斗动画。这使得每个类的代码量都保持在可控范围内，易于理解和修改。

第二点是**便于状态管理**。使用 QStackedWidget 或类似的机制可以方便地在这些视图之间切换，从而管理整个应用的 UI 状态。例如，游戏开始时显示 StartView，点击“开始”后切换到 ShopView。

第三点是实现了**解耦**。例如：BattleView 的职责是“展示”战斗，而不是“执行”战斗。它通过查询 BattleEngine 来获取战斗事件并播放动画。这种设计将“慢速”的动画播放与“快速”的逻辑计算解耦，确保了即使动画效果复杂，也不会影响核心战斗逻辑的执行效率。

三、成员分工

成员	工作内容
李云浩	主导项目主题确立，统筹分工安排，指定环境配置方案，保障项目有序推进 负责部分项目文档的撰写工作 完成 player 类的初步构建，实现玩家相关操作逻辑 开发商店 GUI 模块，实现前后端数据交互与功能对接
施宋杰	承担战斗引擎与商店模块的底层框架搭建及核心代码编写 完成开始页面、结算页面等界面的 GUI 设计与实现
杨司晨	承担战斗引擎与商店模块的底层框架搭建及核心代码编写 扩充宠物、食物与技能种类，完善相关功能调用接口
张天佑	完善宠物类、食物类底层代码并完成具体功能实现 开发商店底层代码，搭建无 GUI 的基础可运行商店页面 编写 main 函数，实现事件驱动框架及基于页面栈的页面切换机制 开发开始界面、战斗界面底层代码，实现界面文本内容的展示逻辑
伍休	优化环境配置方案，指定 GitHub 仓库提交规范，编写项目构建所需的 CMakeLists 文件 搭建宠物类底层框架并完成核心代码编写，实现宠物工厂批量创建宠物的功能 设计并实现宠物技能的底层代码逻辑
陶嘉熙	完善项目初期整体设计文档 优化 player 类代码，新增多项功能并补充关键函数的实现 完成最终程序设计文档的撰写和修改
郑逸彬	对游戏素材库进行解包，提取游戏图像资源 开发战场 GUI 模块，实现宠物战斗的可视化展示功能。
朱凯文	视频制作 负责项目最终汇报视频的制作工作