**南 开 大 学**

高级语言程序设计2-2C++大作业

中文题目： 飞行棋游戏

外文题目： Flying Chess Game

学 号： 2412769

姓 名： 颜浩轩

年 级： 2024级

专 业： 工科试验班（信息科学与技术）

学 院： 计算机学院

完成日期： 2025-5-13

**摘 要**

本文以毕业论文的格式撰写了一款基于 Qt 技术开发的飞行棋游戏程序。该系统不仅实现了经典飞行棋游戏的核心功能，还支持联机多人游戏的扩展。文章通过提出问题、分析现状，明确了飞行棋游戏程序开发的重要性。在技术选型方面，阐述了选择 Qt 构建图形用户界面的原因。此外，详细介绍了系统的整体结构设计、关键功能实现以及在开发过程中遇到的技术难点和解决方案。研究结果表明，基于 Qt 的飞行棋游戏系统能够稳定运行，具有良好的用户交互体验和扩展性，为传统棋类游戏的数字化提供了新的解决思路。

**关键词**：Qt ；C++ ；飞行棋游戏；联机多人游戏；图形用户界面；系统设计；编程语言选型

**ABSTRACT**

This paper presents a Qt-based flight chess game program designed as a graduation project, written in the format of a thesis. The system not only replicates the classic functionalities of the traditional flight chess game but also incorporates online multiplayer features. The study identifies the significance of developing such a game through problem definition and analysis of the current state. In terms of technical choices, the paper examines explains the rationale behind choosing Qt for building the graphical user interface (GUI). Additionally, it elaborates on the overall system design, implementation of core functionalities, and the technical challenges encountered during development along with their resolutions. The results demonstrate that the Qt-based flight chess system operates stably, providing an excellent user interaction experience and scalability, thus offering a novel approach to digitizing traditional board games.

**Key Words**: Qt,C++, flight chess game, online multiplayer, graphical user interface, system design, programming language selection

**目 录**

[一、引 言 1](#_Toc165817278)

1.1 [设计的背景及意义 1](#_Toc165817279)

[1.2 当前研究现状 1](#_Toc165817280)

[1.3 本文的组织结构 2](#_Toc165817282)

[二、技术选型 3](#_Toc165817283)

[2.1 各编译语言的特点 3](#_Toc165817284)

[2.2 组织架构 6](#_Toc165817286)

[三、系统设计 9](#_Toc165817288)

[3.1 整体结构 9](#_Toc165817289)

[3.2 模型设计 10](#_Toc165817290)

[四、系统实现 12](#_Toc165817298)

[4.1 飞行棋规则介绍 12](#_Toc165817299)

[4.2 飞行棋棋盘映射 12](#_Toc165817300)

[4.3飞行棋逻辑实现 15](#_Toc165817304)

[五、总结与展望 17](#_Toc165817311)

[5.1研究总结 17](#_Toc165817312)

[5.2 不足与改进方向 17](#_Toc165817313)

[参考文献 18](#_Toc165817314)

[致 谢 19](#_Toc165817315)

一、引 言

1.1 设计的背景及意义

随着计算机技术的迅猛发展，传统桌面游戏逐渐向数字化、网络化方向转变。飞行棋作为一种经典的家庭棋类游戏，因其简单易学、富有趣味性而深受各年龄层玩家的喜爱。然而，传统的实体飞行棋在交互性和可扩展性方面存在一定的局限性。基于此背景，开发一款基于Qt技术的飞行棋游戏程序，不仅能够复现经典游戏的核心玩法，还能通过网络功能实现多人联机对战，极大地提升用户体验和游戏的互动性。

本项目的设计与实现具有重要的现实意义。首先，通过数字化飞行棋游戏，可以打破地域限制，让玩家在不同地点也能进行实时对战，增强了游戏的社交属性。其次，基于Qt构建图形用户界面，不仅提高了程序的可视化效果，还为进一步的功能扩展和优化提供了良好的技术基础。此外，本项目还为学生提供了一个全面实践面向对象编程、图形界面设计、网络通信等多方面技术的机会，具有较高的教学和研究价值。

1.2 当前研究现状

近年来，随着互联网和移动设备的普及，许多传统棋类游戏已经被开发成数字化版本，并在各种平台上广泛应用。例如，国际象棋、围棋等棋类游戏的在线对战平台已经相当成熟，但国内流行的飞行棋在数字化方面的研究和应用相对较少。现有的飞行棋游戏多为单机版本，缺乏多人联机对战功能，且在界面设计和用户体验上存在一定的不足。

在技术选型方面，C++作为一种广泛应用的编程语言，其跨平台性和丰富的图形界面库（如Qt,EasyX）使其成为开发桌面游戏的理想选择。然而，基于Qt的飞行棋游戏开发案例较少，相关的研究和实践也相对有限。现有的多玩家在线棋类游戏多基于Web或移动平台开发，利用现代网络技术和框架，如WebSocket、React等，与Qt相比，研究方向和技术路径存在较大差异。

综上所述，基于Qt的飞行棋游戏开发不仅填补了这一领域的研究空白，还为探索传统桌面游戏的数字化转型提供了新的思路和方法。

1.3 本文的组织结构

本文共分为五章，结构安排如下：

**第一章 引言**：介绍了本项目的设计背景及意义，分析了当前相关领域的研究现状，并概述了本文的组织结构。

**第二章 技术选型**：详细介绍了项目中使用的各类编程语言及其编译器的特点，阐述了Qt中各种图形用户界面（GUI）库的选择理由，以及各类开发框架的特点和优劣。

**第三章 系统设计**：从整体架构的角度，描述了系统的整体设计结构，重点介绍了模型-视图-控制器（MVC）模式下各部分的关系和联系，确保系统的模块化和可维护性。

**第四章 系统实现**：具体描述了系统的实现过程，包括飞行棋游戏规则的实现、用户界面坐标化的处理以及具体逻辑部分的开发。重点讨论了在开发过程中遇到的技术难点及其解决方案。

**第五章 总结与展望**：对整个项目进行了总结，回顾了设计与实现的过程，评估了系统的功能和性能。同时，指出了项目的不足之处，并对未来的改进方向和可能的扩展提出了展望。

二、技术选型

2.1 各编译语言的特点

在本项目的技术选型过程中，对多种编程语言进行了深入对比分析，包括C、C++、Java、Python、C#和JavaScript等，这些语言因其不同的设计理念和技术特点，适用于不同的开发场景。

C语言作为一种经典的过程式编程语言，以其高效的运行性能和对硬件资源的精细控制能力而闻名。它在操作系统开发、嵌入式设备和高性能计算领域中占据重要地位。C语言的特点包括简洁高效、跨平台能力强以及对底层资源的直接访问。然而，C语言的缺点也很突出，例如缺乏对面向对象编程（OOP）的支持以及对内存管理的严格要求，这使得开发人员容易犯错，如出现内存泄漏和缓冲区溢出等问题。在需要快速开发复杂应用程序或处理高级逻辑的场景中，C语言的使用难度较高。

C++是在C语言基础上扩展而来的，增加了对面向对象编程、泛型编程以及标准模板库（STL）的支持，使其成为开发复杂系统和应用程序的强大工具。C++保留了C语言高效的性能，同时通过类、继承、多态等特性提高了代码的可维护性和复用性。C++的多线程支持和对硬件的直接控制使其非常适合开发需要高性能和低延迟的应用，例如游戏引擎和实时系统。。

Java是一种面向对象的编程语言，以其跨平台能力（“编写一次，到处运行”）和丰富的标准类库而广受欢迎。Java通过Java虚拟机（JVM）实现了良好的可移植性，适用于桌面应用、企业级应用以及分布式系统的开发。它的垃圾回收机制自动管理内存，减少了开发人员手动内存管理的负担。Java还拥有成熟的GUI开发框架，如Swing和JavaFX，然而，与C++相比，Java的运行效率稍逊，这主要是由于JVM的中间解释和运行机制导致的。

Python是一种解释型的高级编程语言，以其简洁的语法和快速开发能力而闻名。Python的代码可读性极高，开发者可以用更少的代码实现同样的功能。它在Web开发、数据分析、人工智能等领域有着广泛的应用，得益于其庞大的第三方库生态。然而，由于Python是解释型语言，其运行效率较低，尤其在处理需要高性能的任务时表现不佳。此外，Python的GUI开发工具（如Tkinter、PyQt）功能有限，相较于C++的Qt等工具，在复杂桌面应用的开发中稍显不足。

C#是由微软开发的一种现代化编程语言，主要用于Windows平台上的应用程序开发。C#结合了C++的高性能和Java的面向对象特性，同时与.NET框架深度集成，提供了强大的开发工具和丰富的库支持。C#的语法设计简洁直观，开发效率高，尤其适合开发企业级应用、游戏（如Unity引擎支持的游戏）以及桌面应用。然而，C#的跨平台支持主要依赖于.NET Core或第三方工具，生态系统对Windows平台以外的环境支持有限，这可能会影响开发灵活性。

综上所述，不同编程语言各有特点和适用场景。java在性能开销上次于C语言。在本项目中，C++因其多线程支持和对硬件的直接控制，对面向对象编程、泛型编程以及标准模板库（STL）的支持而成为最优选择。C++不仅能够满足飞行棋游戏桌面应用的功能需求，还能通过其成熟的开发生态为项目的扩展和维护提供便利。

2.2 组织架构

MVC（Model-View-Controller）架构是一种经典的软件设计模式，通过将系统分为模型、视图和控制器三个模块，明确了数据管理、界面展示和用户交互的职责分工。在本项目中，MVC架构有助于降低代码耦合度，提升系统的可维护性和扩展性。

在MVC架构中，模型（Model）负责管理游戏的核心数据和规则逻辑。例如，在飞行棋游戏中，模型模块包括棋盘状态、玩家信息以及游戏规则的实现。视图（View）负责将模型中的数据以图形化的形式展示给用户，例如游戏界面上的棋盘和玩家的移动过程。控制器（Controller）充当模型和视图之间的桥梁，处理用户的操作输入并将其转化为对模型的修改，同时触发视图的更新。

采用MVC架构能够确保系统的逻辑清晰，模块之间的职责分明，使得功能的扩展和测试更加高效。例如，在本项目中，若需调整游戏规则，只需修改模型模块而不影响视图和控制器的实现。MVC架构在飞行棋游戏中的应用进一步强化了代码的结构化和可读性。

飞行棋游戏支持联机多人对战功能，这要求系统具备良好的分布式架构设计。本项目采用C/S（Client/Server，客户端/服务器）架构，将系统分为客户端和服务器两部分。客户端负责用户界面的展示和本地操作的响应，服务器负责游戏规则的统一管理和玩家间的数据同步。

在C/S架构中，客户端主要使用MVC模式进行功能模块化设计，确保用户界面和本地逻辑的独立性。服务器则专注于业务逻辑和数据存储管理，例如处理玩家的游戏请求、校验棋子移动的合法性以及维护游戏的全局状态。客户端通过网络协议（如TCP/IP）与服务器通信，将玩家的操作请求发送到服务器，并接收服务器返回的结果，从而实现联机对战功能。

C/S架构的优势在于明确了客户端与服务器的职责分工，便于系统的扩展和维护。例如，增加新的客户端功能（如多语言支持）只需修改客户端部分，而无需大幅调整服务器逻辑。此外，C/S架构还提高了数据的安全性，避免了客户端直接访问核心游戏规则和数据的风险。

相比C/S架构，B/S（Browser/Server，浏览器/服务器）架构是一种基于Web的分布式架构，客户端通过浏览器访问服务器。虽然B/S架构在跨平台和部署方便性上具有优势，但对于需要高交互性和实时性的桌面应用（如飞行棋游戏）而言，其响应速度和图形渲染能力不如C/S架构。另一种常见的分布式架构是P2P（Peer-to-Peer，对等网络），但由于其对节点的稳定性要求较高，且难以实现全局状态一致性，不适合本项目的需求。

综合考虑游戏的实时性和复杂性，本项目最终选择C/S架构。C/S架构与MVC模式的结合，使得系统具备良好的模块化设计，同时能够充分满足联机多人游戏的需求。

三、 系统设计

3.1 整体结构

本项目采用了典型的C/S架构和MVC模式相结合的设计方法，将系统划分为客户端和服务器两部分，各部分内部进一步按照功能模块进行划分。客户端负责用户交互，服务器负责游戏逻辑的核心处理及数据同步。

客户端基于MVC模式进行组织，主要包含三个核心包：Controller、Model和View，分别对应控制器、模型和视图的职责划分。

Controller（控制器）：客户端的控制器主要由GameController类组成，负责管理客户端的整体逻辑。它连接View模块与服务器端，通过调用模型和视图接口，完成用户输入、游戏状态更新及与服务器交互的功能。

Model（模型）：客户端的模型层主要用于存储游戏中间状态，提供基本数据支持。GameState类是客户端和服务器之间数据交换的核心对象，包含棋盘状态、玩家信息等。通过序列化和反序列化，该类实现了客户端与服务器之间的高效数据传递。

View（视图）：视图层负责构建用户界面，主要类包括MainView（客户端主界面）、ControlPanel（玩家操作面板）、ConnectDialog（初始连接设置对话框）和BoardPanel（游戏棋盘渲染）。这些组件通过事件监听与控制器交互，确保用户操作能够实时更新到界面上。服务器的结构设计服务器端则专注于处理游戏核心逻辑和客户端同步，主要包括两个核心包：Model和Server。

Model（模型）：服务器端的模型层通过GameModel类管理游戏的全局状态和规则。该类包含棋盘状态、玩家信息等游戏数据，并提供对规则的校验和状态更新的核心逻辑。

Server（服务器控制）：服务器控制部分包括GameServer和ServerController两个核心类。GameServer作为服务器的主程序，负责监听客户端的连接请求并启动游戏服务线程。ServerController则负责处理与客户端交互的具体业务逻辑，包括接收客户端请求、验证数据、更新游戏状态并返回最新状态。

客户端与服务器通过GameState类传递游戏状态，采用序列化对象进行通信。这种设计确保了数据结构的一致性，同时降低了开发复杂度。服务器接收到客户端的操作请求后，通过ServerController处理请求并更新GameModel中的全局状态。随后，服务器将更新后的GameState返回客户端，客户端通过GameController更新界面。

模块间的关系整体结构呈现出清晰的分层和职责分离：客户端与服务器之间通过TCP/IP协议通信，内部各模块之间通过接口和事件驱动交互。客户端的GameController负责连接服务器，GameModel和GameState在数据传递和状态维护中起到桥梁作用，视图模块确保用户操作的直观性。

3.2 模型设计

本项目的模型设计以服务器端的ServerController为核心，结合GameModel和GameState两大辅助类，共同构建了游戏的主要逻辑处理和状态管理体系。ServerController负责游戏全局逻辑的调度与客户端交互，是服务器端的核心控制器，而GameModel则作为底层支持，提供棋盘状态的初始化和基础操作功能。GameState则用于服务器与客户端之间的状态同步，确保了联机游戏的实时性和一致性。

在服务器端，ServerController负责整体的游戏业务逻辑。它管理客户端的连接、接收并解析操作请求、调用GameModel更新游戏状态，并通过GameState向所有客户端广播最新的棋盘状态。当客户端发送请求时，例如投骰子或移动棋子，ServerController会首先验证该操作的合法性，确保玩家的行为符合游戏规则。如果操作有效，ServerController会通过调用GameModel的方法更新棋盘状态，并根据新状态生成一个GameState对象。随后，该GameState对象被广播到所有连接的客户端，从而实现游戏状态的实时同步。例如，当某玩家的棋子因投骰进入特定格子触发规则变更（如返回机场或飞跃多步）时，ServerController能够动态响应并处理复杂的逻辑变动，同时将结果传递给其他玩家。

GameModel类作为服务器逻辑的底层支撑，主要用于存储和操作棋盘的当前状态。通过GameModel，可以动态管理每个格子的状态，例如添加或移除棋子、检测棋子的位置变化等。其设计以模块化为核心，封装了棋盘初始化、玩家状态管理等功能，为ServerController提供了高效而简洁的接口。例如，GameModel在游戏初始化阶段根据玩家数量动态分配棋盘格子，生成初始的棋盘状态，确保系统能够灵活适应不同玩家人数的场景。此外，当玩家进行关键操作（如碰撞或特殊格子效果）时，GameModel能够快速执行相应的规则校验和状态更新。

GameState类则是客户端与服务器之间的核心数据传递对象。通过实现Serializable接口，GameState能够高效地在网络中进行序列化和反序列化。该类的核心设计是一个键值对形式的映射，tileStates属性用来描述每个格子上当前的棋子状态。服务器在处理玩家操作后，会生成一个新的GameState对象，将更新后的棋盘状态和玩家状态以序列化形式发送给客户端，客户端据此更新界面。通过这种设计，GameState能够在不同系统之间实现高效的数据共享，并确保服务器和客户端之间的状态一致性。

总体来看，本项目的模型设计以ServerController为中心，通过GameModel和GameState的协作实现了完整的业务逻辑处理和状态管理。ServerController统筹处理了客户端请求和服务器逻辑，GameModel提供了稳定的底层支持，而GameState则确保了客户端与服务器的高效通信。通过这种设计，飞行棋游戏能够在多客户端环境下实现实时联机与动态交互，显著提升了用户体验。

四、系统实现

4.1 飞行棋规则介绍

飞行棋是一种经典的棋类游戏，可以由2到4名玩家参与。游戏的棋盘结构由多个功能区域组成，包括机场、起始点、路径和终点。每位玩家以颜色区分，分别对应棋盘的四个角：黄色玩家位于左上角，蓝色玩家位于右上角，绿色玩家位于左下角，红色玩家位于右下角。每名玩家拥有四架飞机，初始时所有飞机都停留在各自对应的机场内。

游戏以回合制进行，每位玩家依次投掷骰子以决定当前回合的行动。当玩家掷出5点或6点时，可以选择将机场中的一架飞机移动到自己的起始点，或者选择让棋盘上的一架飞机按照骰子点数继续前进。此外，掷出6点的玩家将获得额外一次行动机会。如果未掷出5点或6点，则玩家只能选择让棋盘上的一架飞机按照骰子点数前进。

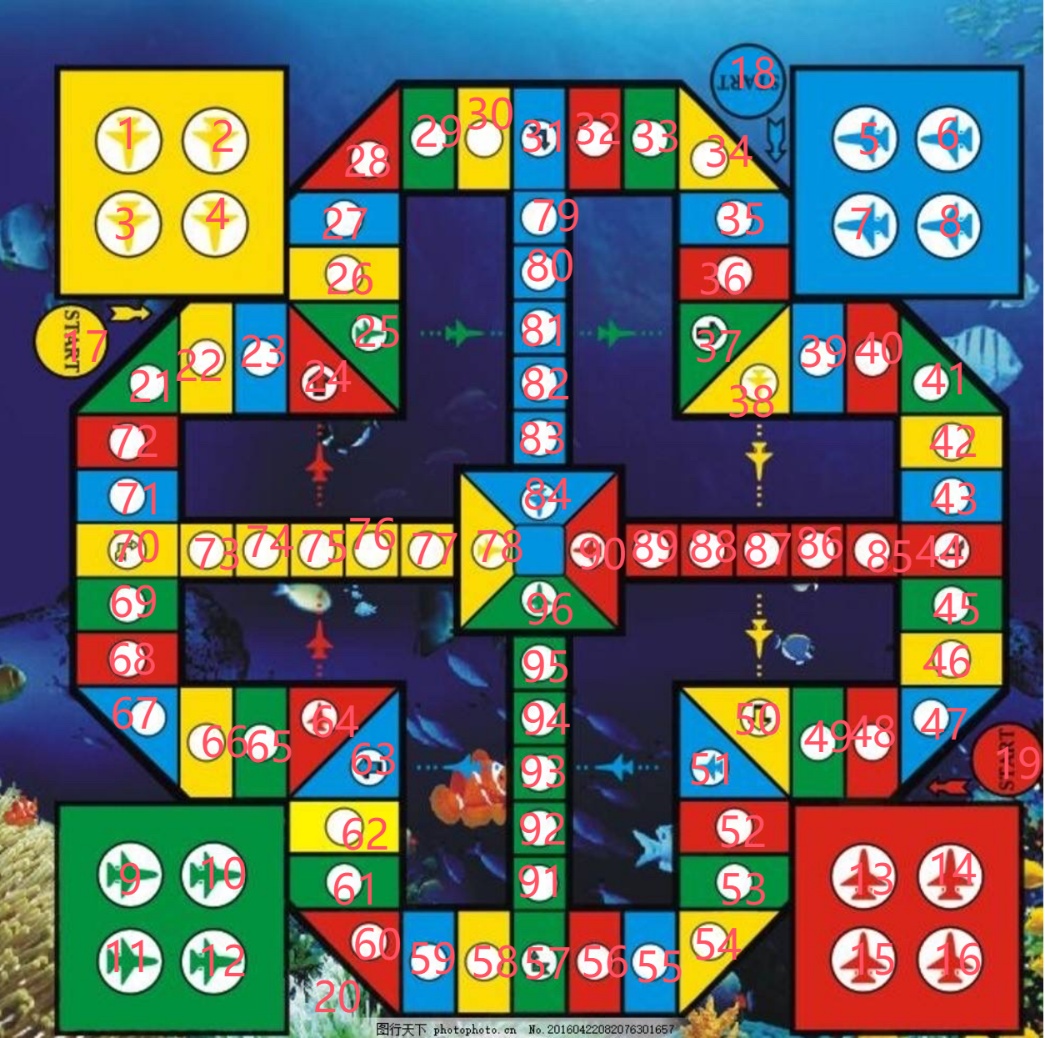
在移动过程中，如果玩家的飞机最终停留在一个已经被其他颜色飞机占据的格子上，则这些其他颜色的飞机将被送回其对应的机场。这一规则增加了游戏的对抗性和策略性。此外，如果玩家的飞机移动后停在环状路径中与自己颜色匹配的格子上，则可以选择额外向前移动四格，从而获得更大的移动距离。

当玩家的飞机到达终点后，标志着该架飞机已完成目标。然而，如果飞机刚好超过终点对应的步数，剩余的步数将会迫使飞机从终点向反方向倒退几格，这种设计增加了游戏的挑战性。游戏的胜利条件是，某玩家的所有四架飞机都成功到达终点，完成棋盘的整个飞行旅程。此时，该玩家宣布获胜。

飞行棋的规则简单易懂，但通过骰子投掷的随机性和移动策略的多样性，带来了极强的趣味性和互动性。这种结合了运气与策略的游戏机制，使得飞行棋在不同年龄段的玩家中广受欢迎。

4.2飞行棋棋盘映射

飞行棋棋盘是游戏的核心组成部分，承担着玩家棋子移动规则的可视化和实时状态展示的作用。如图1所示，为传统飞行棋棋盘，以及对应棋盘格子编号。在本项目中，棋盘映射通过BoardPanel类实现，采用动态绘制和相对坐标体系，既还原了经典飞行棋棋盘的布局，也增强了界面的灵活性和适应性。棋盘结构包括机场、起始点、路径和终点，每个区域通过颜色和形状进行区分。玩家的机场位于棋盘的四个角，分别以黄色、蓝色、绿色和红色表示，起始点则以三角形的形式位于机场外部，与各自颜色对应。路径由棋盘的主要部分组成，呈十字形延伸并环绕棋盘，包含不同颜色的格子以标示各玩家的行进路线。终点区域靠近棋盘中央，排列为一条直线，专为对应颜色的玩家设计。



**图1 传统飞行棋棋盘编号**

棋盘的实现基于相对坐标设计，通过动态计算格子的布局实现自适应窗口大小的功能。在BoardPanel类中，每个格子（Tile）被抽象为一个独立的实体，包含编号、形状、颜色等属性，并能动态显示其包含的飞机状态。棋盘中心作为基准点，各区域的格子位置相对计算而得。例如，机场格子沿对角线分布，路径格子按固定方向排列，起始点和终点则与对应颜色对齐。格子以矩形或三角形的形式绘制，其布局和尺寸随着窗口大小的变化而动态调整。

棋盘状态通过QMap<int, QList<int>>数据结构存储，其中键为格子编号，值为该格子上的飞机列表。BoardPanel类提供了updateBoardState方法，可实时更新格子的状态并触发重绘，将最新的游戏信息呈现给玩家。每个格子不仅会展示其所属的颜色，还会根据飞机数量和类型进行动态标记。通过这种方式，棋盘能够反映出当前的游戏状态，例如某个玩家飞机的移动位置、格子上的飞机数量以及因碰撞或特殊规则导致的状态变化。

BoardPanel的绘制方法注重功能区域的区分和视觉效果的表现。每个格子按照其类型和功能分配不同的底色，并通过边框和编号强化可辨识性。玩家的飞机以圆形表示，在同一格子上的多个飞机则通过排列或叠加的方式显示。棋盘的动态绘制和实时状态绑定使得游戏的每一次操作都能在界面中快速、直观地反映出来，从而增强了玩家的参与感和操作反馈。

如图2所示，本项目的棋盘设计不仅精准复现了飞行棋的经典布局，还充分利用了现代编程技术的灵活性，实现了动态适应性和实时交互性。无论是在不同分辨率的屏幕上运行，还是在复杂的游戏逻辑下处理多种状态变化，该棋盘映射方案都表现出了良好的鲁棒性与扩展性，为游戏的核心体验提供了坚实保障。

图片包含 图示

描述已自动生成

**图2 本项目棋盘编号**

4.3飞行棋逻辑实现

本项目的飞行棋逻辑主要集中在服务器端，通过ServerController类实现了游戏的核心规则和客户端交互功能。该类结合GameModel和GameState两大辅助类，构建了完整的游戏流程，包括玩家操作处理、游戏状态管理和多客户端的联动。

游戏逻辑的基础是服务器端对客户端请求的处理机制。每位玩家通过客户端发送指令到服务器，服务器通过handleClientAction方法解析指令内容并根据当前游戏状态决定后续逻辑。比如，当玩家发送READY指令时，服务器将记录该玩家已准备就绪的状态，待所有玩家准备完毕后，调用initGameAndStart方法初始化棋盘状态并开始游戏。该方法通过GameModel生成初始棋盘状态，并通知第一个玩家开始操作。

在操作过程中，服务器的handleClientAction方法动态处理多种类型的指令。例如，当玩家投掷骰子并选择一架飞机时，服务器解析指令后调用do\_Plan\_OP方法完成棋子移动逻辑，依次检查飞机的当前位置、投掷点数是否合法，并更新棋盘状态。同时，如果玩飞机落点上有其他颜色的飞机，则触发handleCollision方法，将送回其对应的机场。对于特殊情况（如玩家落点为相同颜色的格子或到达终点），服务器也会通过方法进行处理，确保游戏规则的完整性。

服务器在处理操作后，会实时更新GameState对象，并通过broadcastGameState方法将新的游戏状态广播到所有客户端。GameState对象存储了棋盘中每个格子的状态，包括格子上的飞机列表。客户端接收到该对象后，会通过视图组件更新棋盘界面，从而使所有玩家能够同步看到最新的游戏状态。

游戏逻辑还包括对特殊规则的支持。例如，当玩家掷出6点时，除了移动飞机外，还可以获得额外一次操作机会；如果玩家选择跳跃功能，则服务器会调用do\_fly方法计算跳跃路径并更新状态。此外，服务器通过check\_is\_win方法定期检查是否有玩家达成胜利条件。

总体而言，飞行棋的逻辑实现围绕ServerController展开，通过与GameModel和GameState的紧密结合，实现了完整的游戏规则解析、玩家操作处理和状态同步功能。该设计不仅满足了游戏的基本需求，还为未来扩展提供了良好的架构基础，例如可以轻松添加新的规则或功能格子。同时，实时的状态同步机制增强了玩家之间的互动体验，使游戏过程更加流畅和公平。

5总结与展望

5.1研究总结

本项目以经典飞行棋游戏为基础，利用C++语言及其相关技术栈，成功开发了一款支持多玩家联机的飞行棋系统。项目从需求分析出发，采用了C/S架构与MVC模式相结合的设计，充分发挥了两种架构在分层管理和网络交互中的优势。在实现过程中，选用了Qt作为图形用户界面的实现工具，通过动态棋盘绘制和事件驱动机制，实现了对棋盘的精确映射与实时更新。服务器端逻辑则以ServerController为核心，结合GameModel和GameState等模块，实现了对游戏规则的完整解析和多客户端的状态同步。整个系统的运行流程流畅、功能完备，能够为玩家提供良好的交互体验，同时也展现了对经典游戏规则的忠实复现和创新性提升。

5.2不足与改进方向

尽管项目取得了一定的成果，但在开发和运行过程中，仍然存在一些不足之处需要进一步优化与完善。

首先，界面设计仍较为基础。界面的视觉效果在现代应用中显得略为简单，缺乏更高级的动画效果与交互设计。未来可以考虑引入更现代化的UI框架或者加入懂哈，以提升用户体验。

其次，网络通信机制存在一定局限性。当前系统的多玩家联机功能主要基于单线程监听和对象序列化机制，适用于小规模的玩家群体。当玩家数量增加或网络环境较差时，可能会出现性能瓶颈或延迟问题。改进方向可以是引入异步非阻塞通信框架或优化服务器的负载处理能力。

另外，规则扩展性尚未完全优化。当前的游戏规则基于经典飞行棋的基本逻辑，未对新规则或自定义棋盘支持进行深入扩展。未来可以通过引入插件化设计，允许用户自定义棋盘布局、特殊格子功能或玩家数量，以满足更多样化的需求。

总体而言，本项目为飞行棋游戏的数字化和联机化提供了一个可行的解决方案，并为后续优化和功能扩展打下了良好的基础。未来可以在用户体验、性能优化和规则创新等方向上进一步改进 。

**致 谢**

在此，我想对在我程序制作过程中给予帮助和支持的所有人表示诚挚的感谢。首先我要感谢我的高级语言程序设计老师张海威老师，感谢他在课堂中对我的教导，他的知识的讲授和对我的帮助对于我完成这个课程设计至关重要。其次我要感谢南开大学给我提供的平台和上课的机会，给了我更好的进步的途径。此外我要感谢在我的学习过程中为我提供任何帮助的同学。最后感谢各个开源社区的帮助，让我学习代码有了更宽广的途径。