Qt6课程设计实训项目

—斗兽棋

学生姓名:王新龙

学号:2021051615089

年级:2021级

指导教师:龚伟

专业:软件工程

1. 项目基本介绍
2. 项目环境

斗兽棋是一种双人策略棋类游戏。该项目在manjaro定制系统下，是一个基于 Qt Quick 和 JavaScript 实现的斗兽棋游戏，用git进行仓库管理。

1. 游戏规则
2. . 游戏双方

该游戏采用回合制，分红蓝双方，每方8个棋子，蓝方先行。

1. . 棋子种类与等级

鼠（1级）、猫（2级）、狗（3级）、狼（4级）、豹（5级）、虎（6级）、狮（7级）、象（8级）。

1. . 移动规则

棋子可以在竖直和水平方向上移动一格（90单位距离）。

特定棋子（虎和狮）可以跳过河流。

鼠可以进入河流，其他棋子不能进入河流。

1. . 吃子规则

高级棋子可以吃掉比它低级或同级的棋子，但不能吃比它级别高的棋子。

唯一例外是鼠可以吃象。

棋子不能吃位于河中的鼠，鼠在河流中也不能吃其他棋子。

处于陷阱中棋子可以被任何棋子吃掉。

1. . 胜利条件

一方的棋子进入对方的巢穴中。

对方棋子全部被吃掉。

1. 项目需求
2. 功能性需求

（1）. 棋盘布局

棋盘分为7行9列，总共63个格子。

棋盘中央有一块河流区域，分别位于棋盘的第3行、第4行和第5行的第2、3、7、8列。

（2）. 棋子设置

共有16个棋子，红蓝双方各8个棋子，棋子类别与等级如下：

鼠（1级）、猫（2级）、狗（3级）、狼（4级）、豹（5级）、虎（6级）、狮（7级）、象（8级），棋子初始位置需要固定。

（3）. 棋子移动

鼠标点击棋子后，再点击目标位置，实现棋子移动。

棋子只能在横竖方向移动，每次移动一格，棋子不能进入河流。（特殊情况如虎和狮可以跳河）

（4）. 吃子规则

棋子只能吃与自己等级相同或低于自己的棋子（除鼠可吃象外）。

棋子不能吃位于河中的鼠，鼠在河流中也不能吃其他棋子。

处于陷阱中棋子可以被任何棋子吃掉。

吃子成功后，红蓝双方的轮次切换。

（5）. 胜利条件

其中一方棋子进入对方巢穴。

对方的所有棋子全部被吃掉。

游戏结束后显示胜利画面，并提供“继续游戏”和“退出游戏”的选项。

（6）. 游戏轮次

必须实现红蓝双方轮次的切换。

棋子移动成功或者吃子成功后，轮次+1。

1. 非功能性需求

（1）. 游戏界面

棋盘界面需要清晰呈现。

使用图片和颜色区分红蓝双方棋子。

在游戏进行中，用户可以看到当前回合数。

提示用户哪方玩家取得胜利，并提供“继续游戏”和“退出游戏”的选项。

（2）. 响应时间

棋子的移动和交互必须无卡顿，实时响应用户点击操作。

（3）. 可用性

游戏过程中确保不出现异常BUG或崩溃。

所有用户操作应有可预见的结果。

（4）. 可移植性

游戏应可以在不同的操作系统上运行（如Windows、Linux）。

1. 其他需求

（1）. 项目文件组织

确保所有代码文件有合理的文件结构。

代码命名规范，注释清晰，方便他人理解和维护。

（2）. 可扩展性

游戏规则的实现应考虑未来可能的需求变更或新增规则。

设计应考虑不同屏幕尺寸或分辨率的自适应。

（3）. 测试与调试

对移动逻辑、吃子逻辑和胜利条件进行全面测试，确保无误。

进行异常情况测试，如快速点击、多次点击、无效点击等，确保程序稳健性。

（4）. 文档需求

项目应有开发文档，说明各模块功能和实现方式。

游戏说明文档，简述游戏规则和操作方法。

1. 项目详细设计
2. 系统架构

系统主要由三个部分组成：

（1）. 界面层：负责展示棋盘和棋子的界面，处理用户的交互操作。

（2）. 逻辑层：负责棋子的移动规则、吃子规则、胜利条件判断等核心逻辑。

（3）. 数据层：负责存储棋盘状态、棋子状态等数据。

1. 模块设计

（1）. 棋盘模块

初始化棋盘：设置棋盘的初始状态，包括河流和巢穴位置。

更新棋盘：根据棋子的移动或吃子操作，更新棋盘状态。

（2）. 棋子模块

初始化棋子：设置每个棋子的初始位置和属性。

移动逻辑：实现棋子的移动规则。

吃子逻辑：实现棋子的吃子规则。

状态更新：根据游戏进展，更新棋子的状态。

（3）. 用户交互模块

点击事件处理：处理用户点击棋盘和棋子的操作。

界面更新：根据棋盘和棋子的状态，更新用户界面。

（4）. 胜利判断模块

判断胜利条件：判断当前是否有玩家获胜。

显示胜利界面：显示相应的胜利或继续游戏的界面。

1. 方法设计

（1）. 棋盘初始化方法

initChessboard(Chessboard)

输入：棋盘对象

输出：初始化后的棋盘对象

（2）. 棋子初始化方法

initChess(Chess, x, y)

输入：棋子对象、棋子初始位置 x 和 y

输出：初始化后的棋子对象

（3）. 棋子移动方法

move(Chess, Chessboard, mouseX, mouseY)

输入：棋子对象、棋盘对象、鼠标点击位置 mouseX 和 mouseY

输出：更新后的棋子对象和棋盘对象

（4）. 吃子方法

eatChess(eated, eating, chessboard)

输入：被吃棋子对象、吃子棋子对象、棋盘对象

输出：更新后的棋子对象和棋盘对象

（5）. 胜利判断方法

vectory(chessboard)

输入：棋盘对象

输出：胜利状态（0：游戏继续，1：蓝方胜利，2：红方胜利）

1. 数据设计

（1）. 棋子数据结构

Chess {

property int level // 棋子等级

property bool gamer // true代表蓝方，false代表红方

property double middleX: 0 // 记录当前位置的中心点x坐标

property double middleY: 0 // 记录当前位置的中心点y坐标

property bool state: false // 棋子状态，true为可移动，false不可移动

}

（2）. 棋盘数据结构

Chessboard {

property int round: 0 //回合数，走一步加一

property var vectorystate; //胜利状态，0：游戏继续，1：蓝方胜利，2：红方胜利

}

1. 用户界面设计

（1）. 棋盘界面

使用 Chessboard.qml 定义棋盘布局。

（2）. 棋子界面

在棋盘界面上进行布局。

使用Chess.qml设置棋子基本属性

（3）. 开始界面

使用 Start.qml 定义开始界面布局。

1. . 胜利界面

在棋盘界面上进行布局。