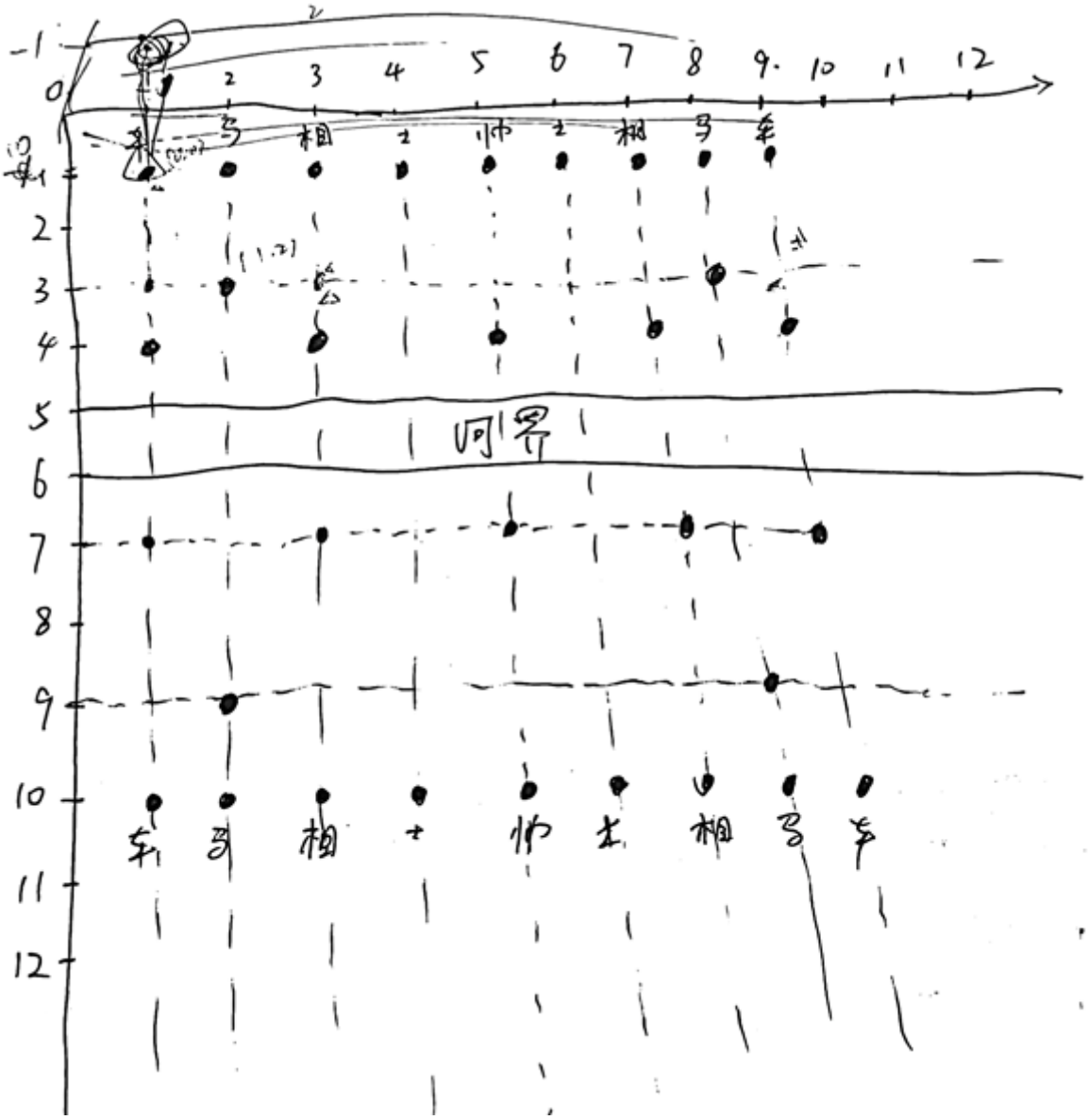
中国象棋游戏技术文档

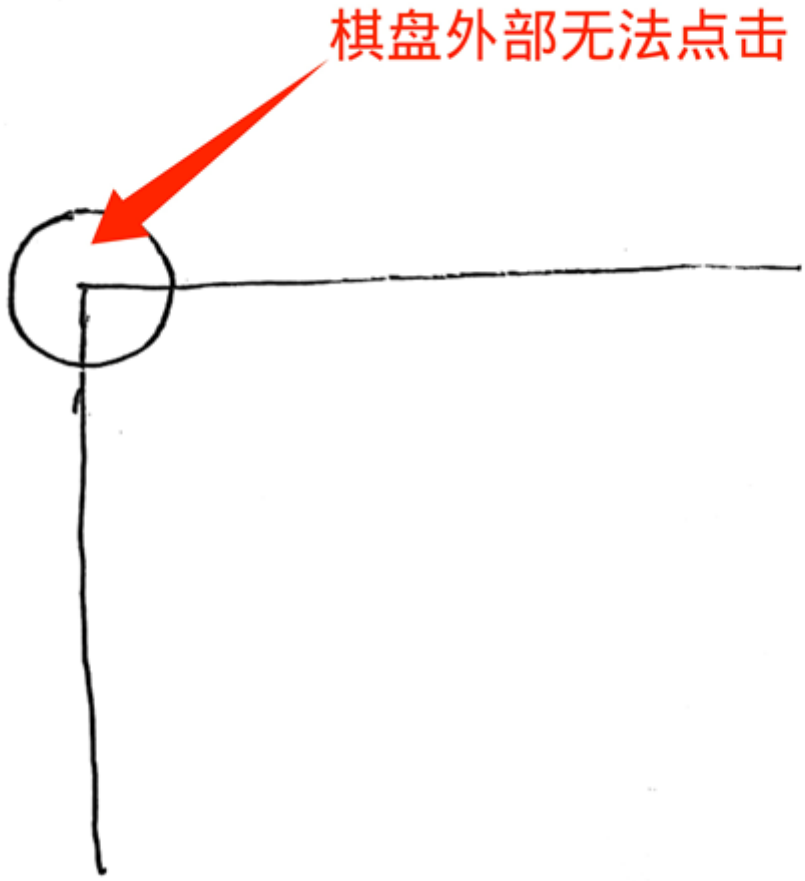
1. 项目概述

中国象棋游戏是一款基于Qt Quick框架开发的跨平台象棋应用程序，实现了双人对战（PVP）、人机对战(PVE)和网络对战三种游戏模式。项目采用QML语言构建用户界面，结合C++后端逻辑混合编程完成完整游戏功能。

中国象棋项目游戏采用ui和逻辑分离的开发方法，所有ui用qml语言编写（包括象棋的棋盘和棋子），根据棋盘的坐标特征确定棋子大小以及棋盘的坐标。棋盘坐标如下图所示：



说明：本项目中的棋子相对于棋盘的坐标有变化，坐标（1,1）表示棋盘左上角“车”的坐标，依此类推。这样设置的原因是如果以棋盘（0,0）为棋子的起始坐标，会导致最边缘的棋子无法处理点击事件，因为边缘棋子一半在棋盘外，一半在棋盘内，如下图：



由于时间问题，所以采取移动棋子坐标的方法保证每个棋子的整体都在棋盘内。

1.1. 主要功能

- 双人对战模式：支持本地两位玩家轮流走棋
- 人机对战模式：集成AI引擎，提供不同难度级别的电脑对手
- 网络对战模式：支持通过TCP/IP协议进行网络对战
- 游戏规则说明：详细介绍象棋规则和棋子走法
- 音效系统：背景音乐和多种游戏音效
- 动画效果：棋子移动、吃子、胜利等动画效果
- 设置选项：音乐和音效开关控制

1.2. 技术栈

- 前端：QML、Qt Quick Controls、Qt Quick Layouts
- 后端：
  - **JavaScript**：处理业务逻辑
  - **C++**：实现象棋核心逻辑和AI算法

- 网络：QTcpSocket、QTcpServer
- 多媒体：Qt Multimedia
- 动画：Qt Quick Animations：处理音效和背景音乐

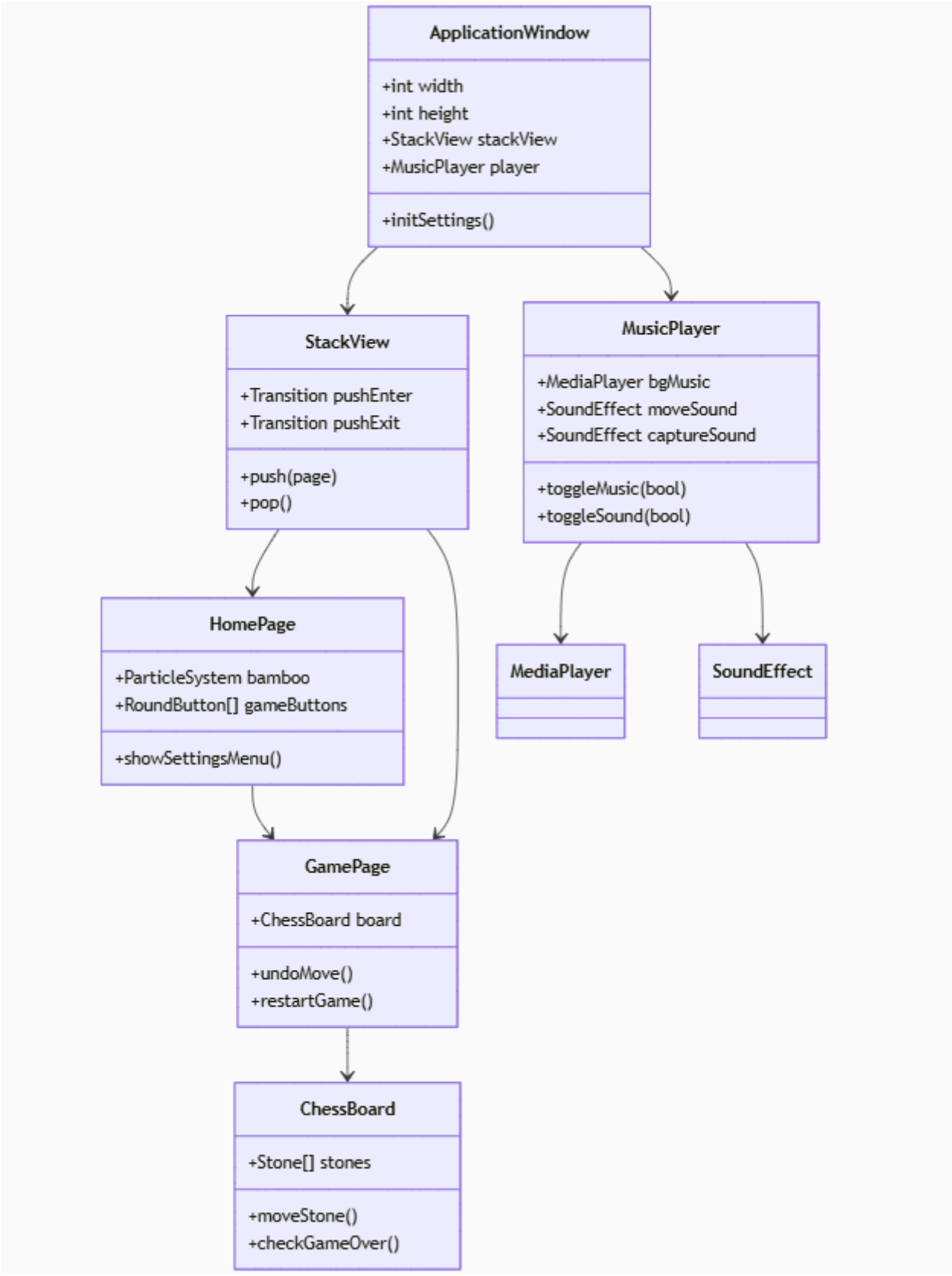
2. 项目结构

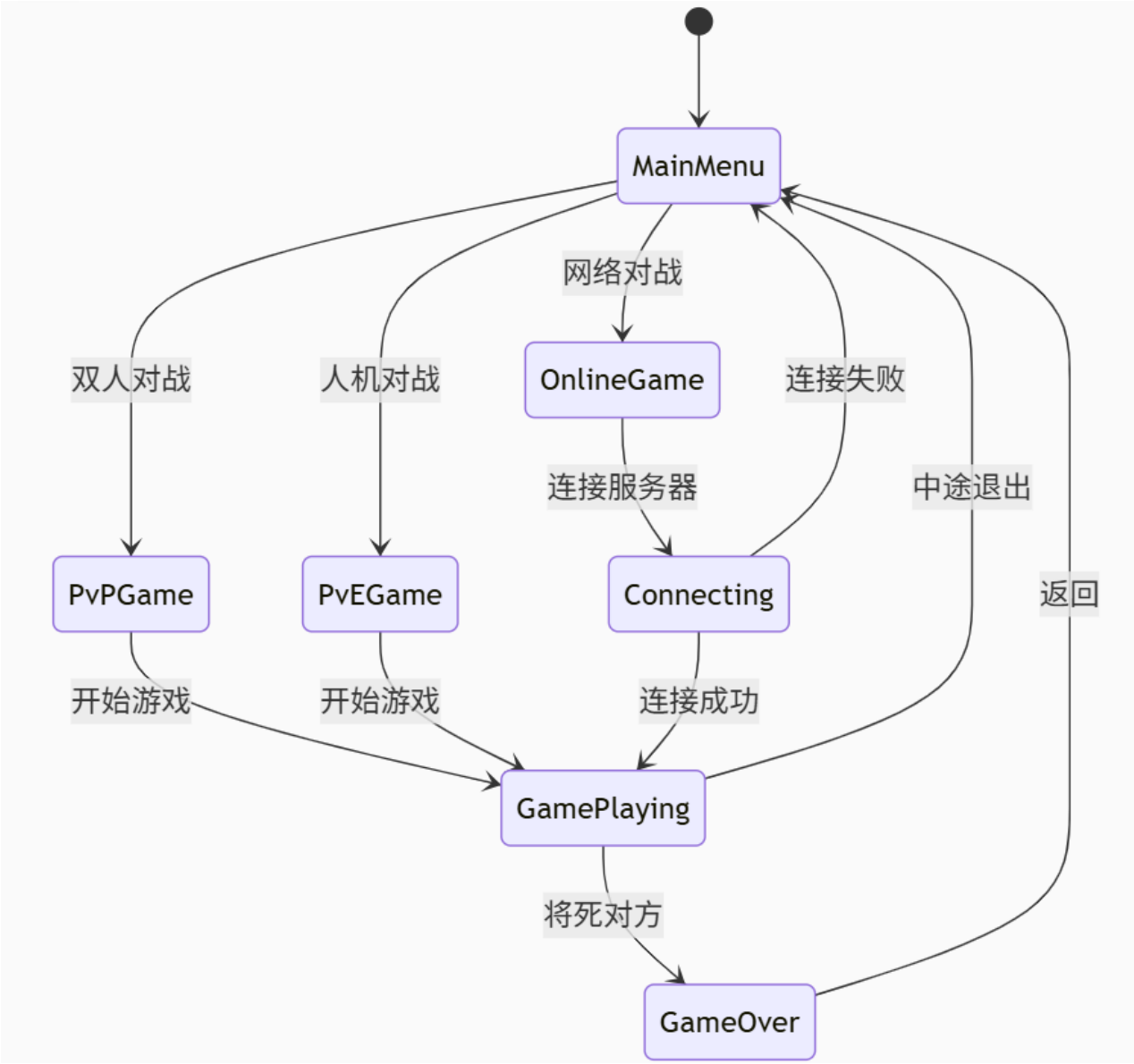
2.1. 主要QML文件

1. `Main.qml`：应用程序入口，初始化窗口和`StackView`
2. `HomePage.qml`：主菜单页面，提供游戏模式选择，背景粒子效果(竹叶飘落)，设置菜单(音乐/音效控制)
3. `ChessBoard.qml`：双人对战棋盘实现
4. `AIChessBoard.qml`：人机对战棋盘实现
5. `NetworkChessBoard.qml`：网络对战棋盘实现
6. `ChessPiece.qml`：棋子组件
7. `MusicPlayer.qml`：音效管理系统
8. `RulesPage.qml`：游戏规则说明页面
9. `AboutPage.qml`：关于页面
10. `PvpGamePage.qml/PveGamePage.qml`：双人/人机对战页面容器

2.2. 核心C++类

1. `Board/AIGameBoard/NetworkBoard`：游戏逻辑核心类
2. `Stone`：棋子数据类
3. `EleeyeEngine`：AI引擎接口类


ApplicationWindow
+int width
+int height
+StackView stackView
+MusicPlayer player
+initSettings()
StackView
+Transition pushEnter
+Transition pushExit
+push(page)
+pop()
MusicPlayer
+MediaPlayer bgMusic
+SoundEffect moveSound
+SoundEffect captureSound
+toggleMusic(bool)
+toggleSound(bool)
HomePage
+ParticleSystem bamboo
+RoundButton[] gameButtons
+showSettingsMenu()
MediaPlayer
SoundEffect
GamePage
+ChessBoard board
+undoMove()
+restartGame()
ChessBoard
+Stone[] stones
+moveStone()
+checkGameOver()



3. 核心组件详细说明

3.1. Main.qml - 应用程序入口

3.1.1. 应用窗口架构设计

- 根元素选择：使用 `ApplicationWindow` 作为顶级容器，提供原生窗口特性（标题栏、尺寸控制等）。
- 动态宽高比：通过 `width: (height/10)*6` 强制保持 6:10 的棋盘比例，确保棋盘显示完整。
- 窗口约束：设置 `minimumWidth/maximumWidth` 等属性限制窗口缩放范围，防止布局错乱。

3.1.2. 导航系统实现

- **StackView** 管理：作为核心导航控制容器。

```
StackView {
    id: stackView
    initialItem: "HomePage.qml"  // 初始页面
    anchors.fill: parent  // 填充整个窗口
    pushEnter: Transition { ... }  // 自定义页面进入动画
    popExit: Transition { ... }    // 自定义页面退出动画
}
```

- 平滑过渡效果：使用 `PropertyAnimation` 实现页面切换时的透明度渐变（150ms 动画时长）。

3.1.3. 全局资源初始化

- 音乐播放器注入：通过 `MusicPlayer { id: player }` 实例化全局音频控制器。
- 启动逻辑：在 `Component.onCompleted` 中调用 `Controller.initial()` 初始化音乐状态，确保应用启动时音效配置生效。

3.2. HomePage.qml - 主菜单页面

作为中国象棋应用的主界面，首先构建了完整的背景体系，通过Image组件加载背景图，并设置透明度为0.3实现朦胧效果，底层Rectangle填充淡米色底色形成协调的视觉层次。顶部的设置按钮采用 `ToolButton` 实现，定位在窗口左上角，通过Menu组件展开包含音乐控制、音效开关和退出选项的下拉菜单，菜单项图标根据当前状态动态切换，如音量图标会在静音时自动变为禁用状态。

通过 `ColumnLayout` 实现垂直排列。五个 `RoundButton` 分别对应不同游戏模式，采用统一的样式规范：宽度占屏幕50%、固定宽高比1:4、圆角半径5像素，这些样式通过 `butt onStyle` 对象集中管理。每个按钮点击时都会触发两步操作：先播放点击音效增强反馈感，再通过 `StackView` 进行页面导航，同时传递当前窗口尺寸参数，确保棋盘正确适配。

使用粒子系统 `ParticleSystem` 实现竹叶飘落背景效果， `ParticleSystem` 作为管理器， `ItemParticle` 负责渲染，Emitter控制发射行为。粒子使用 `SpriteSequence` 实现三帧动画循环，通过设置不同的 `frameDuration` 使竹叶飘落过程更自然。发射器配置为顶部水平排列，以每秒3个的速率发射，粒子初速度设置向下40单位并带有20单位的水平随机偏移，配合10单位的垂直加速度，模拟竹叶飘落时摇摆下坠的物理效果。

```
ParticleSystem {
    id: _bamboo
    ItemParticle {
        delegate: SpriteSequence {
            // 竹叶精灵动画
        }
    }
    Emitter {
        id: leafEmitter
        emitRate: 3
        lifeSpan: 9000
        // 发射器配置，性能优化：设置 lifeSpan: 9000（9秒生命周期）避免内存堆积，不启用影响器，降低计算开销。
    }
}
```

状态管理方面通过属性绑定实现实时同步。音乐和音效的开关状态既关联到设置菜单的图标显示，又通过 `Controller` 与全局音乐播放器联动。所有交互元素都包含视觉反馈机制，如按钮按下时颜色变深，设置菜单弹出时有轻微弹性动画，提升了界面的响应感和操作质感。

文件最后通过 `Component.onComplete` d完成了初始化工作，确保所有动态元素能正确获取初始尺寸。整个页面采用完全响应式设计，所有尺寸计算都基于父元素宽度，使得在不同分辨率的设备上都能保持一致的视觉效果和操作体验。

遇到的问题：

- 粒子系统性能优化：通过调整发射率和生命周期平衡效果与性能
- 按钮在不同屏幕尺寸下的适配：使用相对尺寸而非固定像素值

3.3. ChessBoard.qml/AIChessBoard.qml - 棋盘实现

构建了一个功能完善的中国象棋游戏界面，通过多种 QML 技术实现了丰富的视觉效果和流畅的交互体验。这两个文件都采用了 Canvas 组件来绘制中国象棋特有的棋盘布局，包括网格线、九宫格内的斜线以及楚河汉界的文字标识。棋盘绘制不仅考虑了基本的线条走向，还特别处理了棋盘中间断开的特殊布局，还原了传统中国象棋的棋盘样式。

在棋子交互方面，这两个文件都实现了完整的棋子选择、移动和吃子逻辑。通过 `TapHandler` 捕获用户的点击事件，结合自定义的点击位置计算算法，能够准确识别用户点击的是哪个棋格或棋子。选中棋子时会触发缩放动画和发光边框效果，给予用户明确的视觉反馈。移动和吃子时则会播放相应的音效，并通过使用 `PropertyAnimation` 来实现平滑的移

动过程。特别是吃子动画采用了圆形扩散效果，对于将军被吃的情况还额外添加了旋转边框和多重扩散圆环的特殊效果。

当游戏分出胜负时会显示一个半透明的黑色遮罩层，上面展示胜利方的信息。胜利文本采用了多层叠加的技术，底层是带有颜色渐变的发光文字，上层则是带有轮廓效果的主文本，通过组合动画实现了淡入、循环缩放和颜色渐变的复合效果。棋盘背景采用了圆角矩形设计，配合图片填充和透明度控制，并通过 `OpacityMask` 确保背景图片也能呈现圆角效果。

两个文件都采用了基于数据绑定的响应式设计，所有 UI 元素的尺寸都基于棋盘单位动态计算，确保了在不同屏幕尺寸上都能保持协调的布局比例。通过 Connections 元素将游戏逻辑信号与 UI 表现相连接，实现了棋子状态变化、游戏结束等事件到视觉效果的自动映射。

遇到的问题：

- 棋子精确定位：需要将屏幕坐标转换为棋盘坐标
- 动画同步问题：确保动画完成后再更新逻辑状态
- 性能优化：大量棋子状态变化时的渲染效率

3.4. AIChessBoard.qml - 人机对战模块

构建了一个完整的人机对战中国象棋系统，通过深度集成的AI逻辑和丰富的视觉交互效果实现了专业级的游戏体验。该文件在基础棋盘功能之上扩展了AI对战模块，通过Timer延迟AI走棋，模拟思考过程，当轮到AI走棋时会启动一个100毫秒的定时器，在延迟结束后调用 `computerMove()` 函数执行AI决策，这种设计既避免了AI立即响应带来的机械感，又保证了游戏的流畅性。同时拥有多级难度控制，通过调整搜索深度和评估函数来实现。AI走棋逻辑与玩家操作共享同一套动画系统，通过 `onComputerMoved` 信号触发移动动画创建流程，确保视觉表现的一致性。

实现了动画效果系统，当检测到将军被吃时，会通过 `createGeneralCaptureEffect` 函数创建三层扩散圆环动画，每个圆环间隔200毫秒依次出现，形成波浪式的视觉效果。这些动画效果通过动态组件技术实现，使用 `Qt.createComponent()` 加载 `AnimationEffect.qml` 模板并传入不同的参数控制位置、颜色和延迟时间。移动动画则采用 `ParallelAnimation` 并行执行缩放和透明度变化，通过easing曲线控制动画的加速度，使整个过程更加自然流畅。

交互系统采用 `TapHandler` 处理用户输入，实现了点击状态管理。点击逻辑区分了多种情况：选中己方棋子、取消选中、切换选中以及吃子尝试等。游戏状态检测机制实时监控棋盘变化，当 `onGameOver` 信号触发时会显示胜利动画界面，这个界面包含多层文本渲染、循环缩放动画和颜色渐变效果，通过 `SequentialAnimation` 和 `ParallelAnimation` 的组合

创造出引人注目的视觉效果。所有视觉元素都采用响应式设计，尺寸基于棋盘单位动态计算，确保在不同分辨率设备上都能保持协调的布局比例。

遇到的问题：

- AI响应速度与用户体验的平衡
- 防止AI思考时用户操作导致的竞态条件
- 悔棋功能的实现与AI状态的同步

3.5. NetworkChessBoard.qml - 网络对战模块

核心逻辑围绕网络对战功能展开，通过多层架构实现复杂的交互流程。文件首先建立网络连接管理模块，采用对话框形式处理主机创建与加入游戏的流程。主菜单对话框作为入口点，提供"创建游戏"和"加入游戏"两个选项，通过`NetworkBoard`组件底层实现TCP/UDP通信。当用户选择创建游戏时，系统初始化服务器监听端口；选择加入游戏则通过IP输入框连接指定主机，连接状态通过`connectionStatus`属性实时反馈到界面。

棋盘交互逻辑是文件的关键部分，通过`TapHandler`处理玩家操作。点击事件首先校验网络连接状态和当前回合权限，防止非操作时段的误触。坐标转换模块将屏幕点击位置转换为棋盘行列坐标，结合棋子选中状态实现三步操作流程：选中己方棋子、移动至目标位置、通过网络发送棋步数据。移动验证由C++端的`ChessEngine`处理，返回结果触发QML端的动画效果。吃子逻辑特殊处理将军情况，通过创建`AnimationEffect`组件实现红色/蓝色波纹扩散特效，视觉上强化关键棋步。

网络通信层通过信号槽机制实现实时同步。当接收到对手移动数据时，onOpponentMoveReceived信号触发本地棋盘状态更新，同时播放对应的音效。聊天系统独立于棋局通信，采用文本协议传输消息，通过TextArea展示对话历史，ScrollView确保消息过多时可滚动查看。胜利判定逻辑与本地模式类似，但额外增加了网络连接状态的校验，防止断线时的错误判定。

视觉呈现方面，棋盘绘制采用Canvas动态渲染，区别于静态图片以实现更好的分辨率适配。九宫格斜线单独使用一个Canvas层绘制，线宽加粗至2.5像素以提升视觉辨识度。"楚河汉界"文字通过Text组件实现。棋子使用Repeater动态生成，每个棋子绑定到`NetworkBoard`的stones模型，通过`modelData`属性获取实时状态，当棋子状态变化时自动触发位置和可见性更新。

状态管理通过多个属性维护关键状态：`connectionStatus`跟踪网络连接阶段，`myTurn`标志当前操作权限，`myColorIsRed`记录玩家颜色分配。这些状态通过属性绑定自动更新UI，如回合提示文本根据`myTurn`动态变化。胜利动画作为独立组件覆盖在棋盘上方，使用`SequentialAnimation`组合透明度变化、文字缩放和颜色循环，营造强烈的胜利反馈。

最后整合所有功能模块，棋盘区域占据主视觉空间，聊天面板固定在底部，状态提示位于顶部，形成清晰的信息层级。所有网络操作封装在`NetworkBoard`组件中，使QML层专注于界面呈现和用户交互。

遇到的问题：

- 网络延迟导致的同步问题
- 黑红方的棋子移动、吃棋等同步问题

3.6. MusicPlayer.qml - 音效系统

功能实现：

- 将各组件内音频资源与settings.js 中状态控制相关联，播放状态音量为80，关闭状态音量为0
- 在主界面设置一个控制按钮，添加相关的设置图片资源，以及相关行为（actions），通过对相关按钮的控制实现音乐音效的播放与暂停。
- 音效的播放是短音效，将相应的音效放在合适的位置，通过一些点击行为实现对音效的控制。

关键技术：

1. 多媒体组件：

```
MediaPlayer {
    id: _bgMusic
    source: "qrc:/sounds/bgm.wav"
    loops: MediaPlayer.Infinite
}
SoundEffect {
    id: _moveSound
    source: "qrc:/sounds/move.wav"
}
```

2. 全局控制：

    - 通过JavaScript控制器管理开关状态
    - 属性绑定实现实时音量调整

遇到的问题：

- 音频播放异常问题

曾出现音频插入后无法播放的情况，经过多次排查和资料查阅，发现是音频文件路径引用错误或格式不兼容导致，后续通过规范资源路径（如统一使用qrc:/格式）和提供多种音频格式备选（如准备WAV 格式），解决了该问题。

- 控制逻辑适配问题

最初尝试用play函数统一控制音乐和音效播放，但发现音效通过play函数控制时，在按钮打开时就会自动播放，不符合场景需求。经过调整，采用音量控制的方式替代直接播放控制，通过动态调节音效音量（如需要时将音量设为 0，触发时恢复正常音量），更好地适配了不同场景下的音效播放需求。

- 功能关联与调试难点

开发中最大的难点是将控制部分（JavaScript 逻辑）与音频播放功能有效关联，以及解决 “逻辑无误但播放失败” 的问题。即使代码逻辑看似正确，仍可能因 QML 组件生命周期、信号绑定时机等细节问题导致播放失败，这就需要耐心调试，逐行排查组件初始化顺序、信号连接状态等。

- 全局控制的妙用

在音乐控制功能未能完美实现时，尝试采用全局控制设置，将音频播放器实例设置为全局可访问，统一管理音乐和音效的播放状态、音量等参数，避免了局部控制导致的冲突和不一致问题，最终证明该方案可行，实现了较为稳定的音频控制效果。

3.7. AboutPage.qml- 开发者页面

该界面是主界面下的一个子界面，主要用于展示与开发者相关的信息，如开发团队介绍、指导老师、版本信息等，方便用户了解该游戏的开发背景和相关信息。

1. 界面布局

资源图片作为背景，设置了暖色调的底色。

- 用垂直布局（ColumnLayout）为主要布局容器，并通过anchors.centerln:parent 使其居中显示。
- 标题部分使用了较大的粗体字体，并设置为居中对齐，采用棕色作为文字颜色，与整体色调相协调，突出了页面主题。
- 标题下方为内容区域，使用 Text 组件展示开发团队介绍、版本信息（如 “版本号：1.0.0”）等内容，根据需要调整文字大小、颜色和行间距。
- 底部设置一个返回按钮（Button），通过点击该按钮返回到主界面，按钮的 text 属性设置为 “返回”，并绑定相应的点击事件处理函数。

2.交互逻辑

- 当用户点击返回按钮时，触发按钮的 clicked 信号，调用主界面提供的接口或方法，实现从 AboutPage.qml 界面切换回主界面。
- 界面加载时，自动显示预设的开发者相关信息，无需用户进行额外操作。

3.8. RulesPage.qml- 游戏规则页面

该界面是主界面下的另一个子界面，其主要功能是向用户展示游戏的相关规则、胜利条件等内容，帮助用户更好的清楚游戏相关操作。

1. 界面布局

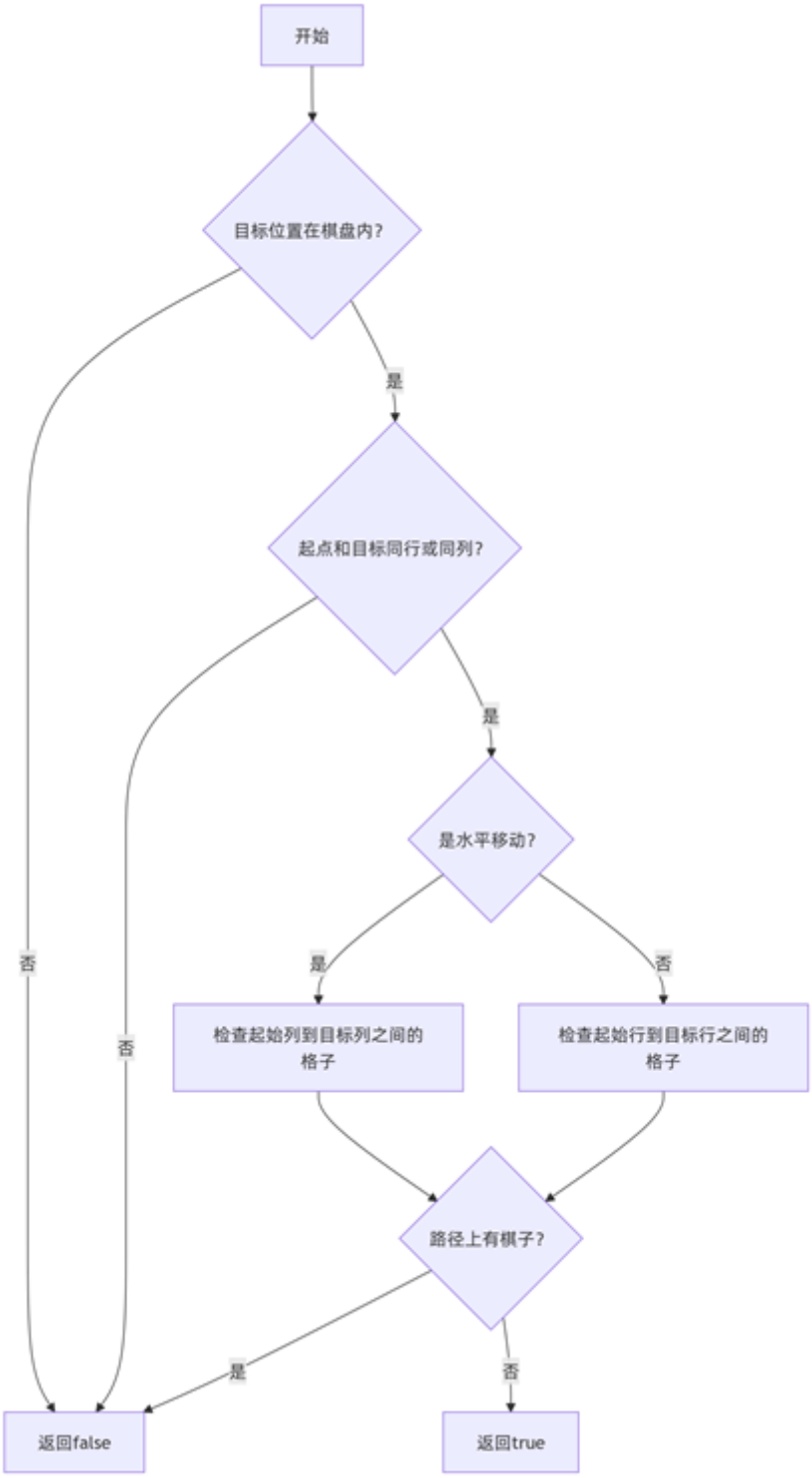
- 采用滚动视图（ScrollView）作为外层容器，以适应规则内容较多的情况，方便用户滚动查看所有规则。
- 在滚动视图内部使用垂直布局（Column）放置规则内容，包括标题和具体规则条目。
- 标题区域使用 Text 组件，设置合适的字体样式和大小，使其清晰可见。
- 具体规则条目使用多个 Text 组件或 ListView 组件（当规则条目较多且有一定规律时）进行展示，每个规则条目设置编号或项目符号，增强可读性。
- 底部设置返回按钮（Button），功能与 AboutPage.qml 中的返回按钮一致，用于返回主界面。

2.交互逻辑

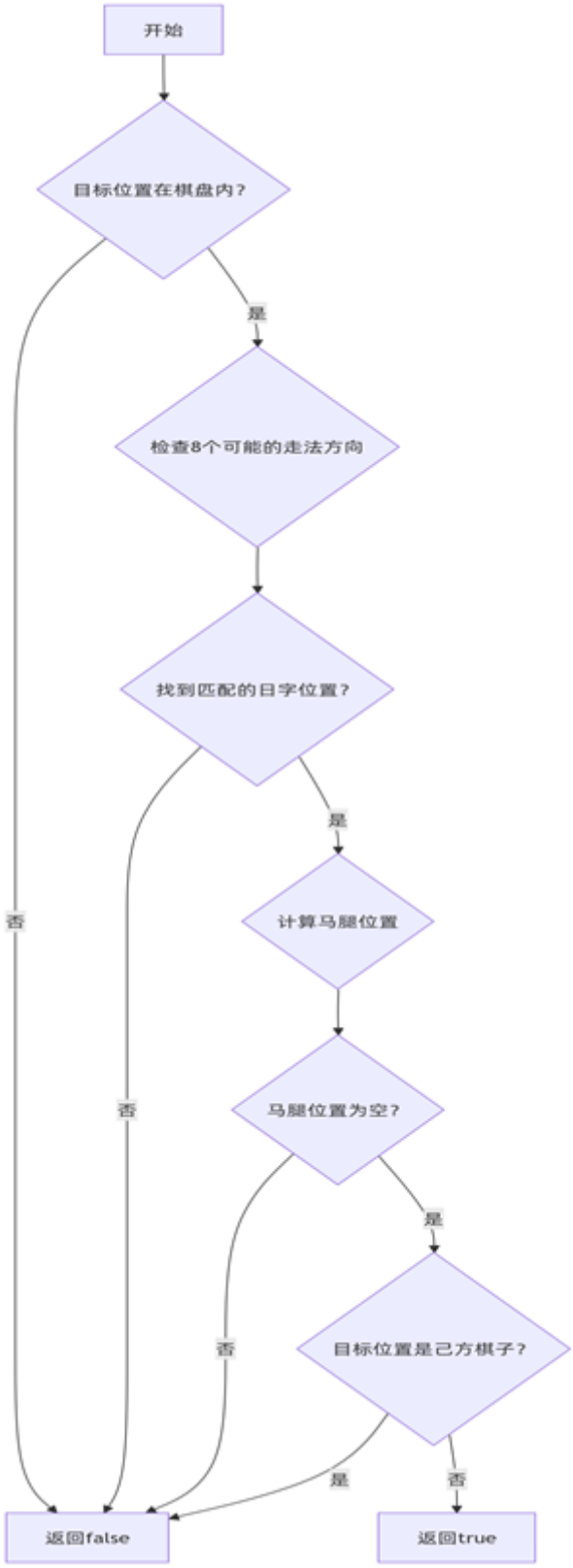
- 用户点击返回按钮时，触发 clicked 信号，执行切换到主界面的操作。
- 当规则内容超过界面显示范围时，用户可通过鼠标滚轮或触摸滑动等方式在滚动视图中上下滚动，查看全部规则内容。

4. 走棋吃棋逻辑

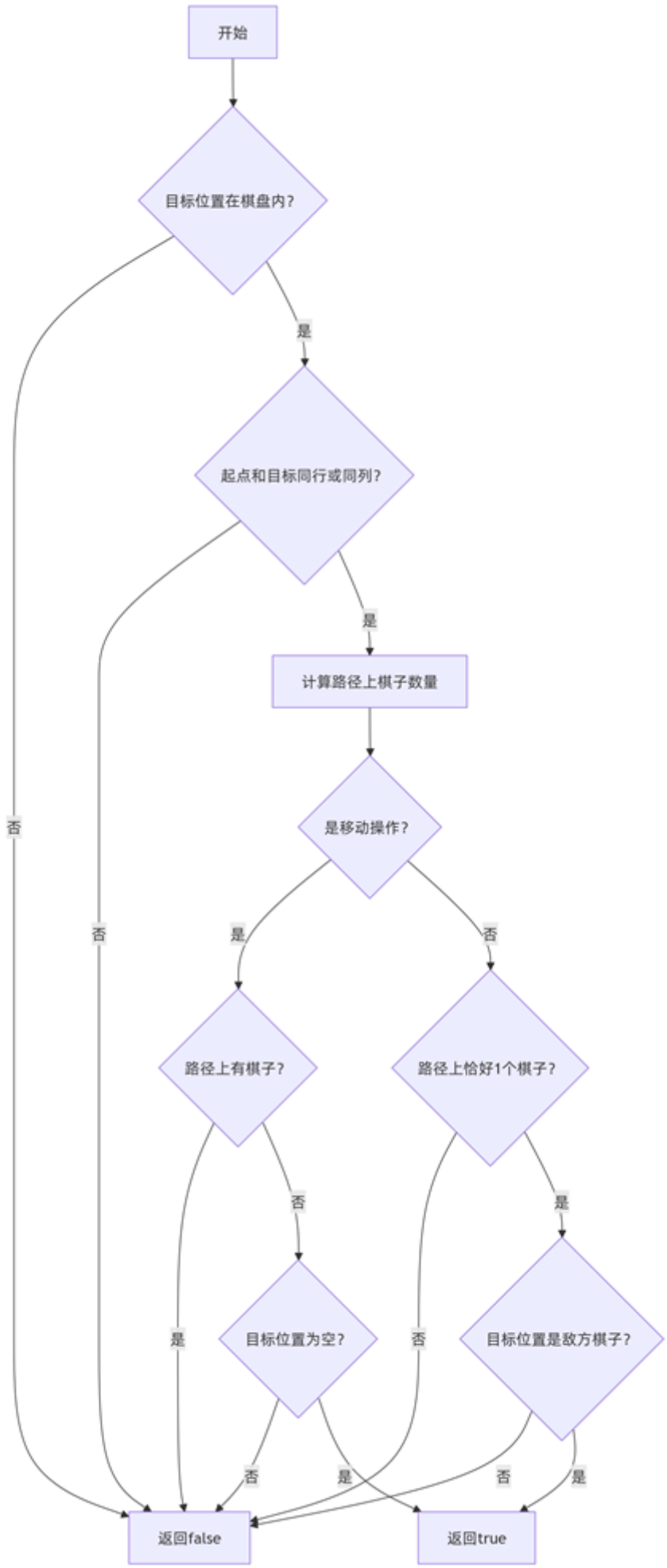
4.1. 车的走/吃棋规则


开始
目标位置在棋盘内?
是
否
起点和目标同行或同列?
是
否
是水平移动?
是
否
检查起始列到目标列之间的格子
检查起始行到目标行之间的格子
路径上有棋子?
是
否
返回false
返回true

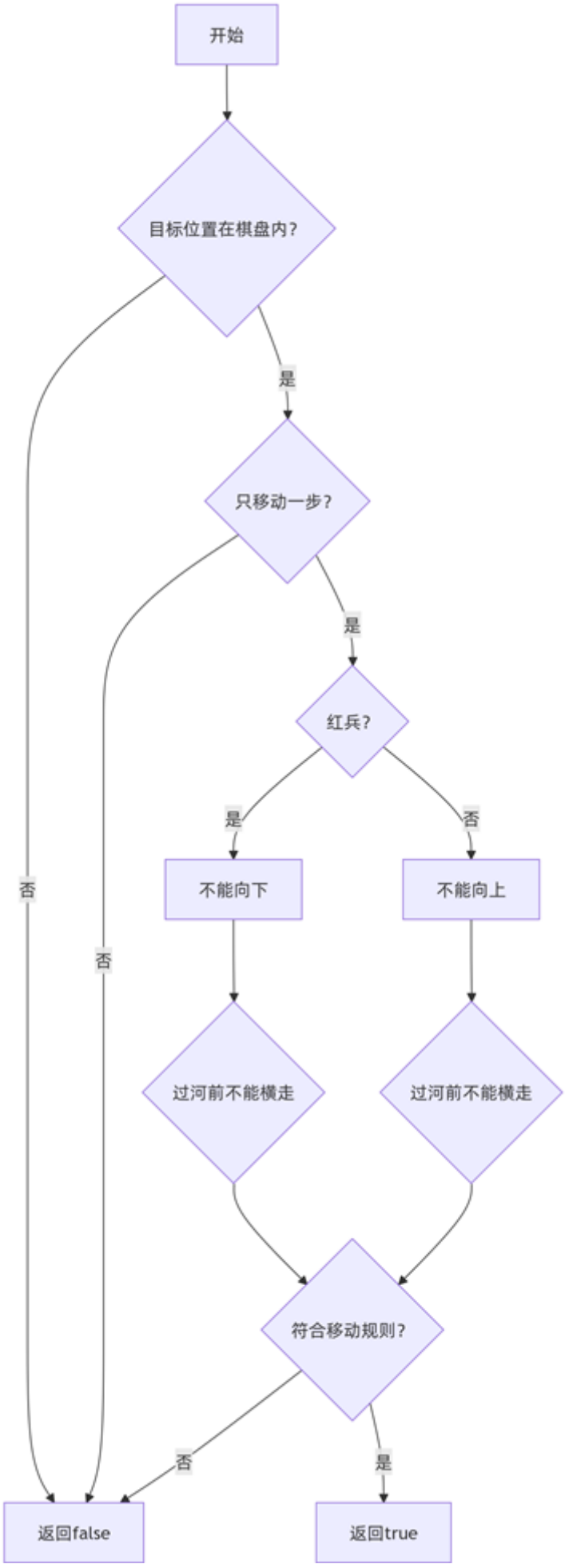
4.2. 马的走/吃棋规则


开始
目标位置在棋盘内？
是
否
检查8个可能的走法方向
找到匹配的日字位置？
是
否
计算马腿位置
马腿位置为空？
是
否
目标位置是己方棋子？
是
否
返回false
返回true

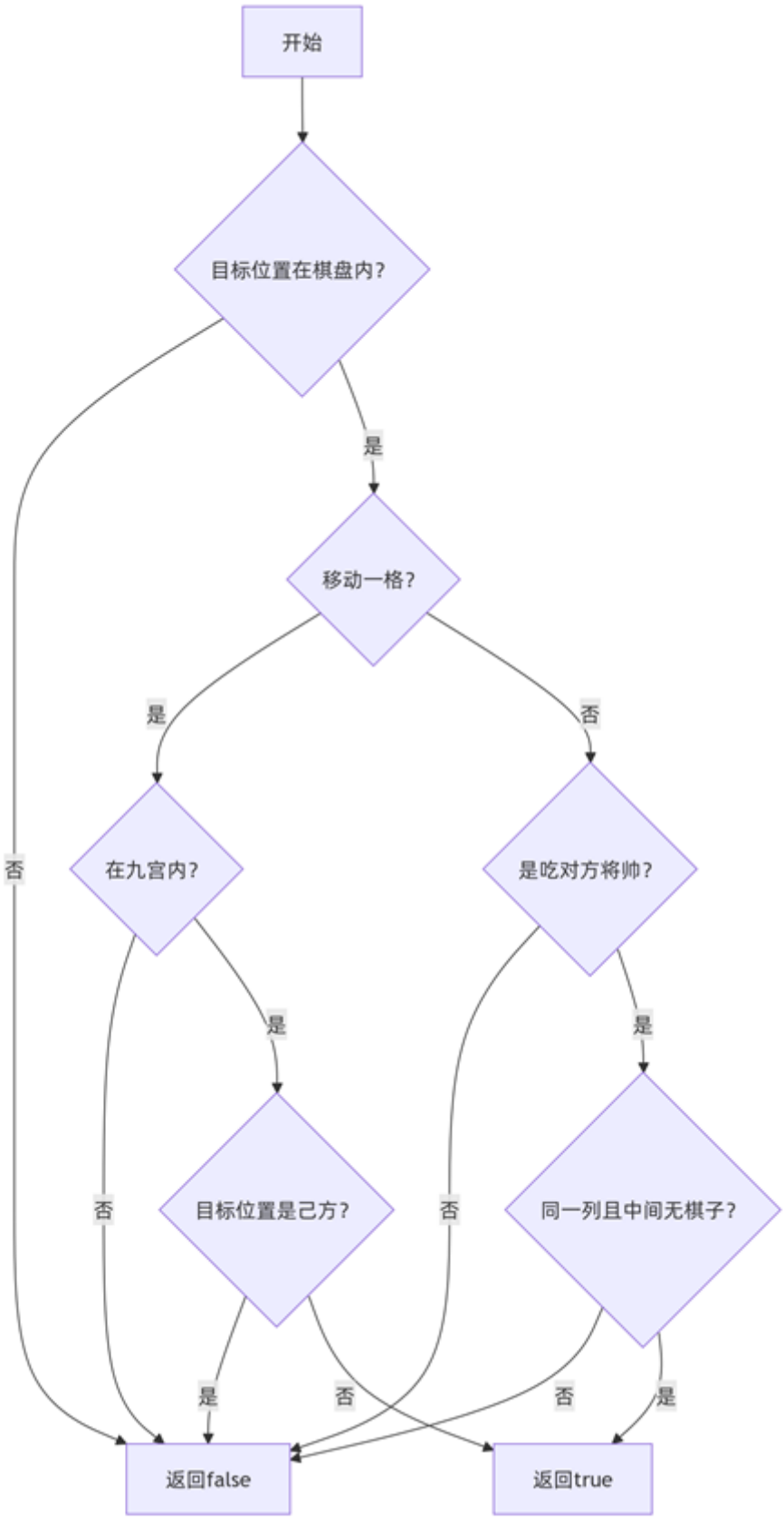
4.3. 炮的走/吃棋规则


开始
目标位置在棋盘内?
否
是
起点和目标同行或同列?
否
是
计算路径上棋子数量
是移动操作?
是
否
路径上有棋子?
路径上恰好1个棋子?
是
否
否
是
目标位置为空?
目标位置是敌方棋子?
是
否
否
是
返回false
返回true

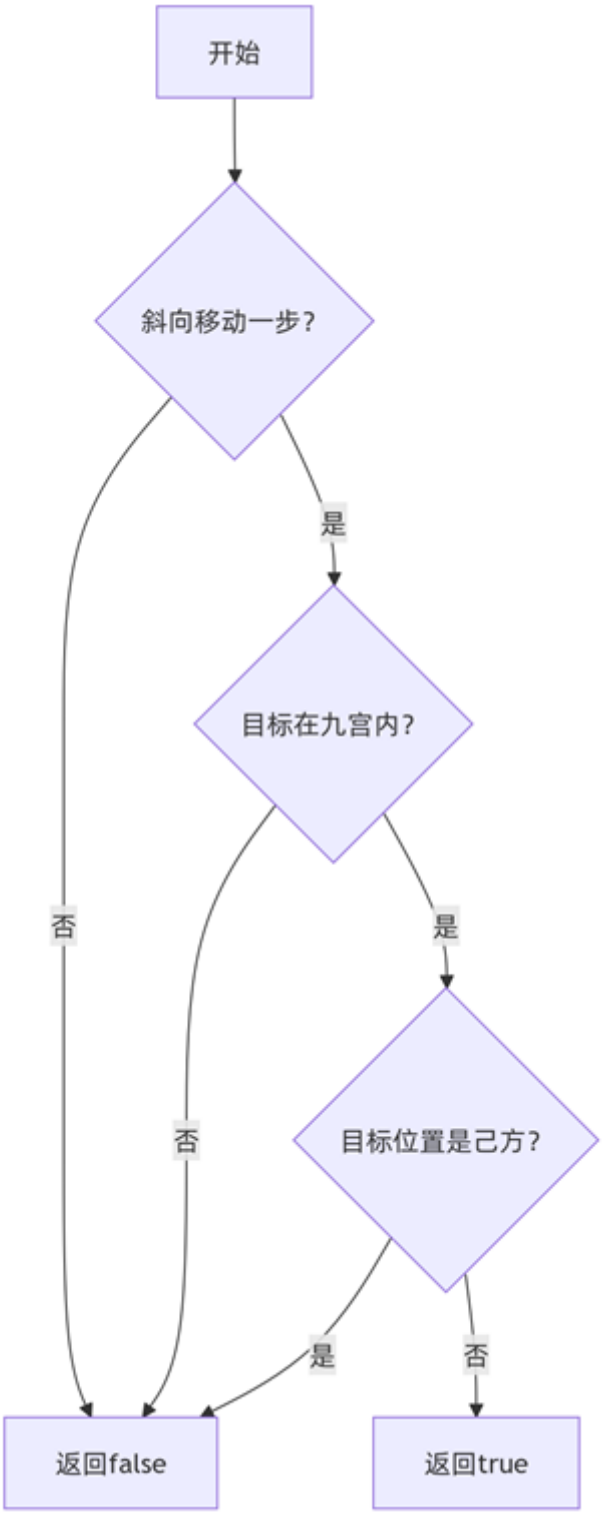
4.4. 兵的走/吃棋规则


开始
目标位置在棋盘内？
否
是
只移动一步？
否
是
红兵？
是
否
不能向下
不能向上
过河前不能横走
过河前不能横走
符合移动规则？
否
是
返回false
返回true

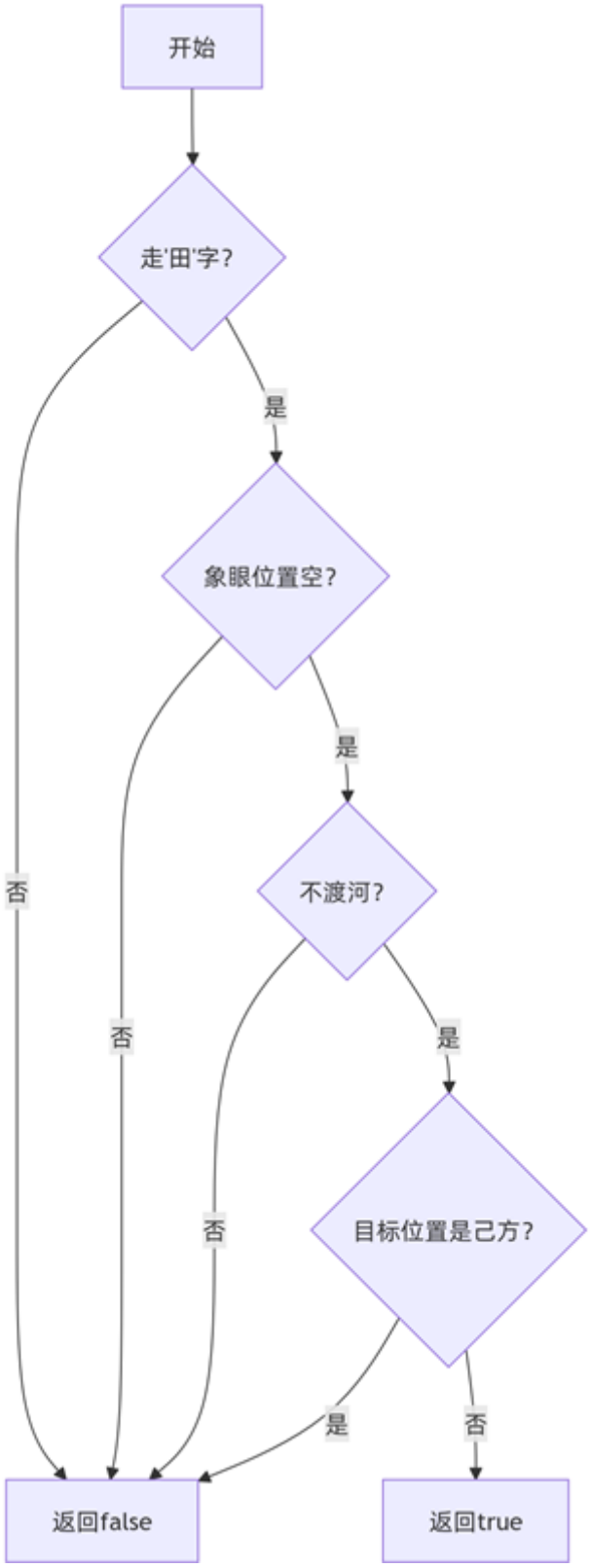
4.5. 将的走/吃棋规则


开始
目标位置在棋盘内?
是
否
移动一格?
是
否
在九宫内?
是吃对方将帅?
否
是
否
是
目标位置是己方?
同一列且中间无棋子?
否
是
否
否
是
返回false
返回true

4.6. 士的走/吃棋规则

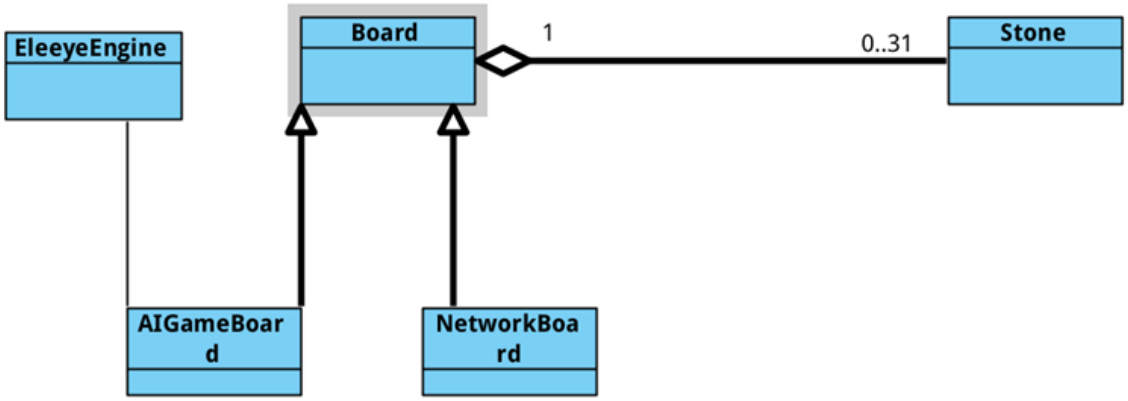


4.7. 相的走/吃棋规则



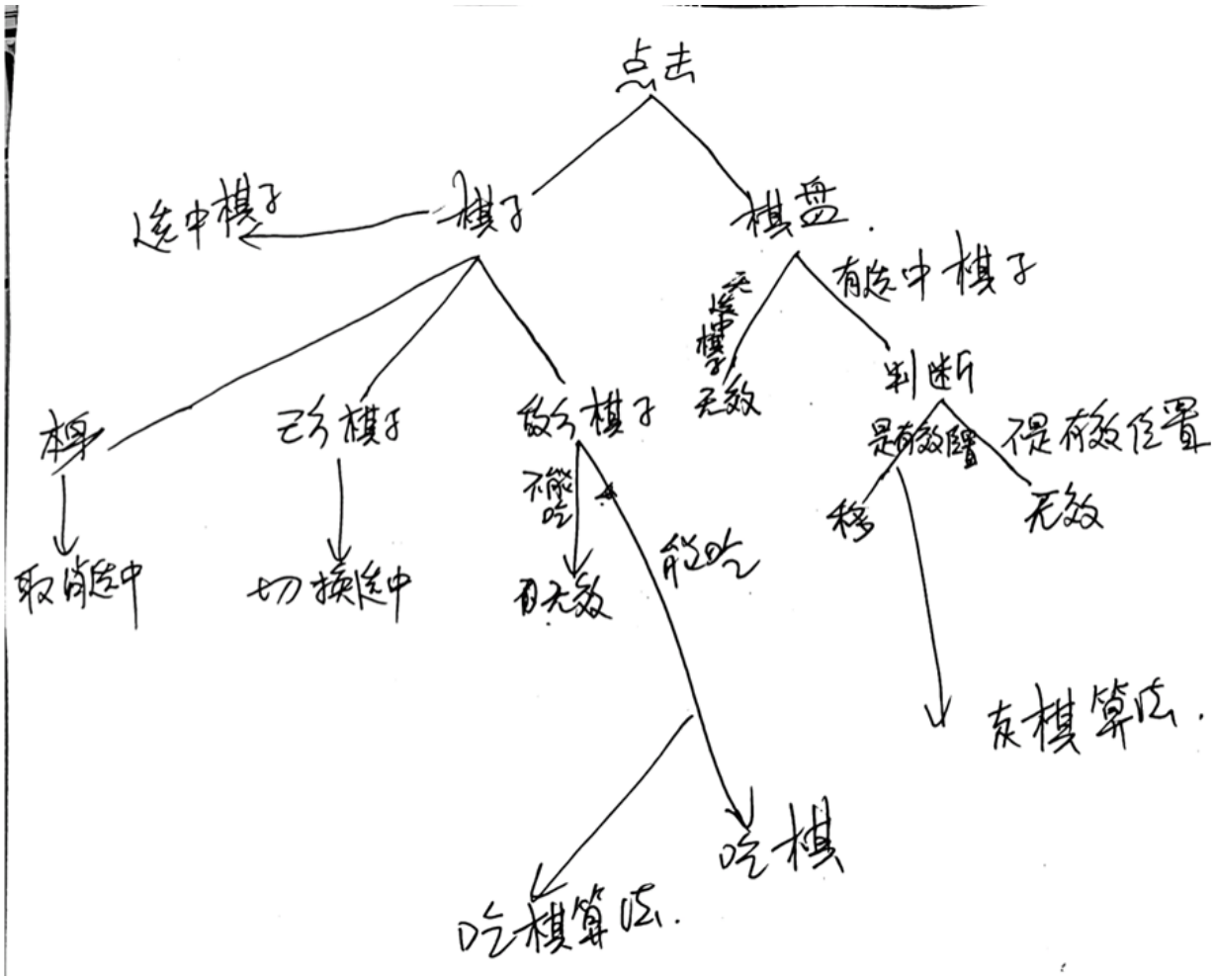
5. 功能实现

5.1. UML类图



5.2. 双人对战

双人对战为对战双方轮流走棋，直到其中一方的将/帅被吃游戏结束，本项目中Board类为游戏的底层逻辑实现类，其中包括各个棋子的走棋方法以及吃棋规则，并且设置一方走棋时，另一方不能进行操作。具体走棋判断流程如下图所示：

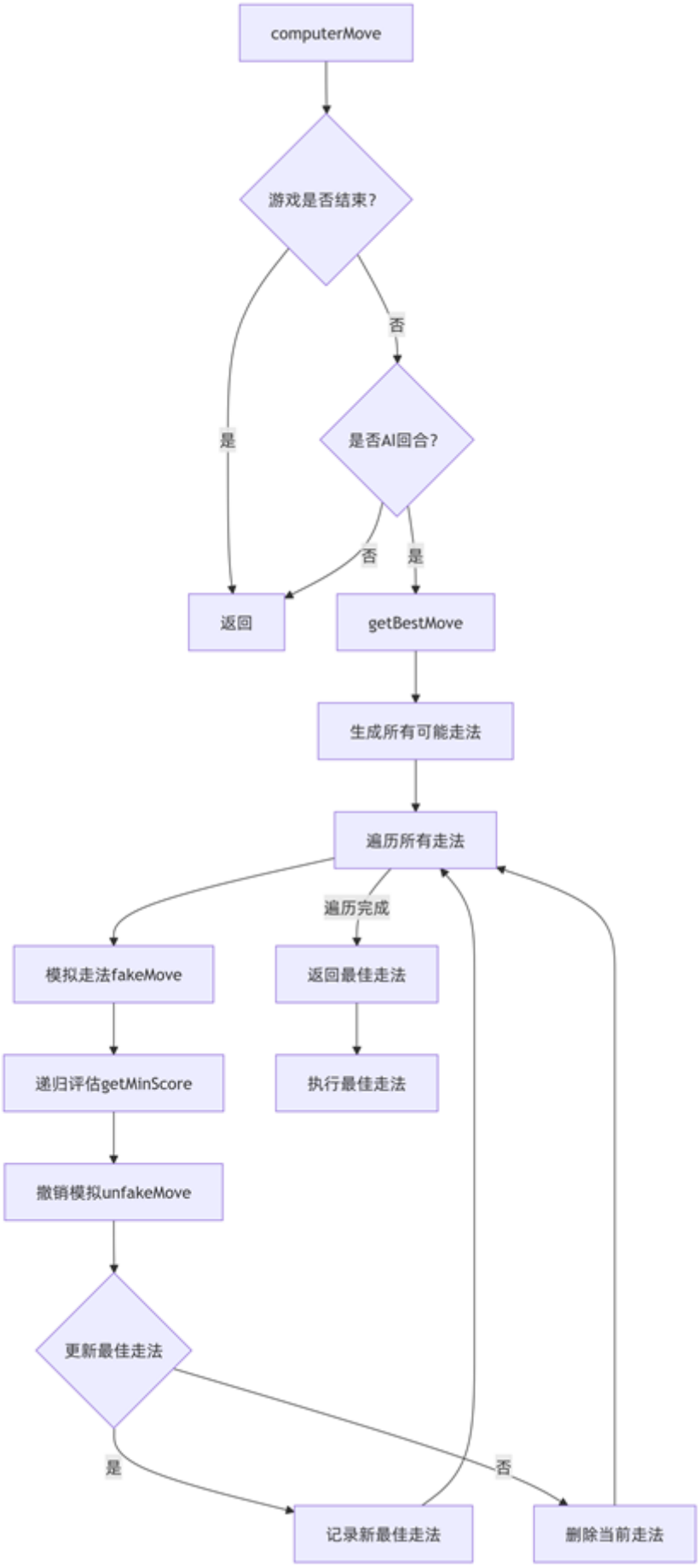


每次点击棋盘会出现多重情况，采用决策树的方法实现走棋。

5.3. 人机对战

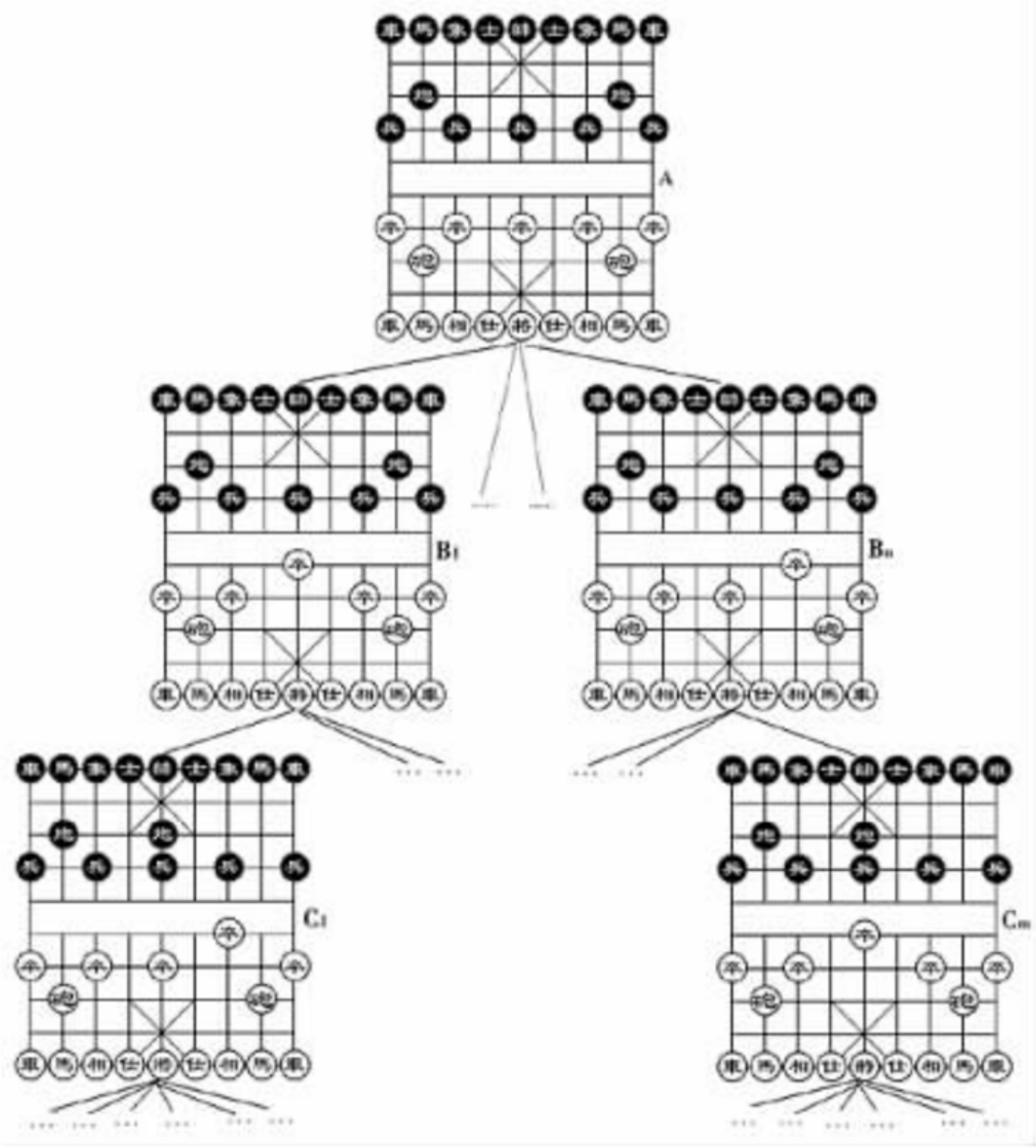
5.3.1. 人机对战的第一个迭代版本

人机对战功能参考了《PC游戏编程（人机博弈）》书籍，该书籍已经放在本项目目录中，同时参考了其他博主的人机博弈算法，本项目基于极大极小算法(Minimax)和Alpha-Beta剪枝优化，通过评估棋盘状态选择最优走法。具体走棋流程如下：


computerMove
游戏是否结束?
是
否
是否AI回合?
否
是
返回
getBestMove
生成所有可能走法
遍历所有走法
遍历完成
模拟走法fakeMove
返回最佳走法
递归评估getMinScore
执行最佳走法
撤销模拟unfakeMove
更新最佳走法
是
否
记录新最佳走法
删除当前走法

附：极大极小算法

设想下象棋的情形，两人对弈，我们将其中一位叫做甲，另一位叫做乙。假定现在该甲走棋，甲可以有40种走法（不论好坏）；而对甲的任一走法，乙也可以有与之相对的若干种走法。然后又轮到甲走棋，对乙的走法甲又有若干种方法应对……如此往复。显然，我们可以依此构建一棵博弈树，将所有的走法罗列出来。在这棵树的根部是棋局的初始局面。根的若干子节点则是由甲的每一种可能走法所生成的局面，而这些节点的 子节点则是由与之相对的乙的每一种可能走法所生成的局面……在这棵树的末梢，是结 束的棋局，甲胜或者乙胜或者是双方都无法取胜的平局。下图为博弈树：

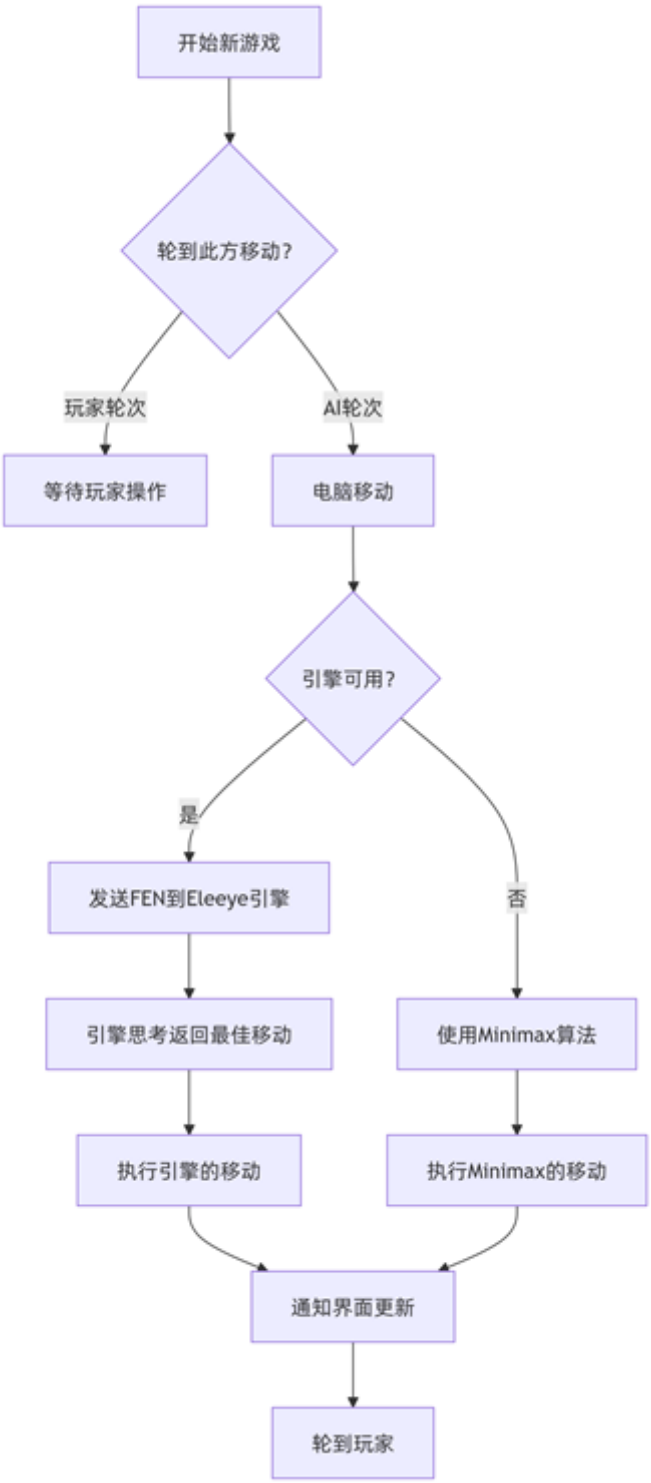


然而，如果遍历整棵树，显然是不可能的。极大极小算法也就是减少遍历次数，评估当前局面的最优走法，我们确定好搜索深度后，用评估函数算出最优分数，同时这样也能设置人机的难度（本项目中搜索深度建议为3,如果大于这个值，会导致程序崩溃，算力消耗太大）。

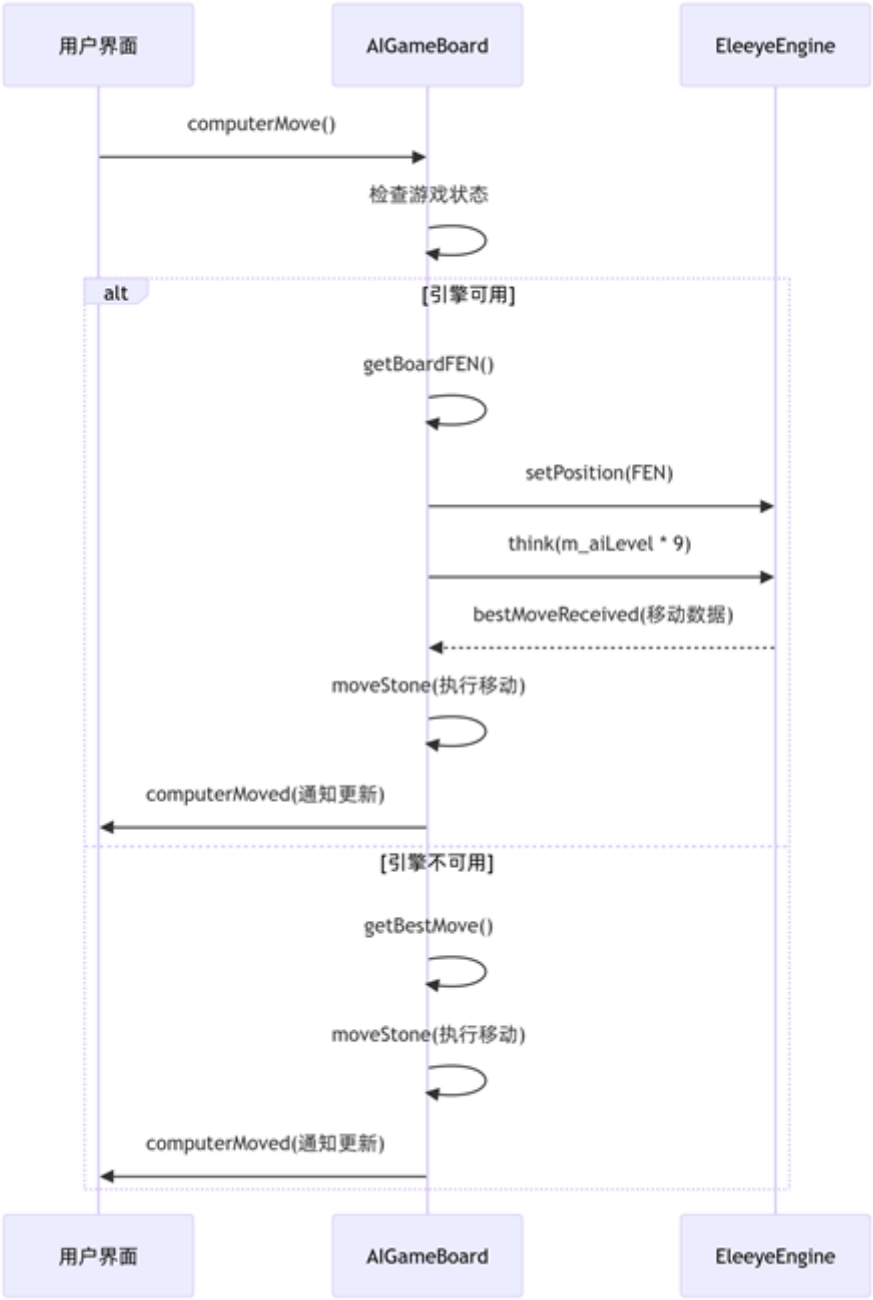
5.3.2. 人机对战的第二个迭代版本

`AIGameBoard` 类是一个基于 `EleeyeEngine` 实现人机对战功能的棋盘类，通过使用 `Eleeye` 象棋引擎提供强大的 AI 功能，同时实现了备用的 `Minimax` 算法。该类继承了基本的 `Board` 类，增加了电脑玩家操作、AI 难度控制、引擎交互等功能。（`Eleeye` 游戏引擎开源地址**https://github.com/xqbase/eleeye.git**）

1.功能实现流程图



2.电脑移动核心逻辑



3.核心功能说明

- AI 引擎通信

引擎初始化：在构造函数中创建并启动 `EleeyeEngine`

最佳移动处理：通过 bestMoveReceived 信号接收引擎计算结果

思考深度控制：aiLevel 属性影响引擎思考时间

FEN位置描述：将当前棋盘状态转换为标准的 FEN 格式

- Minimax 算法（引擎不可用时备用）

博弈树构建：递归遍历所有可能的移动路线

评价函数：根据棋子价值和位置评分

剪枝优化：使用Alpha-Beta剪枝提高效率

分层递归：getMinScore/getMaxScore 实现层次结构

- 棋子移动管理

虚拟移动：fakeMove/unfakeMove 允许在不实际改变棋盘状态下进行推演

移动合法性检查：依赖基类Board的走法规则实现

移动封装：Step类封装移动的所有相关信息

4. **ELEEYE**游戏引擎使用方法（该引擎已经放在项目目录的根目录下）

```
cd eleeye-master/eleeye  # 进入引擎源码目录
chmod +x makefile.sh     # 添加执行权限
./makefile.sh          # 运行编译脚本
```

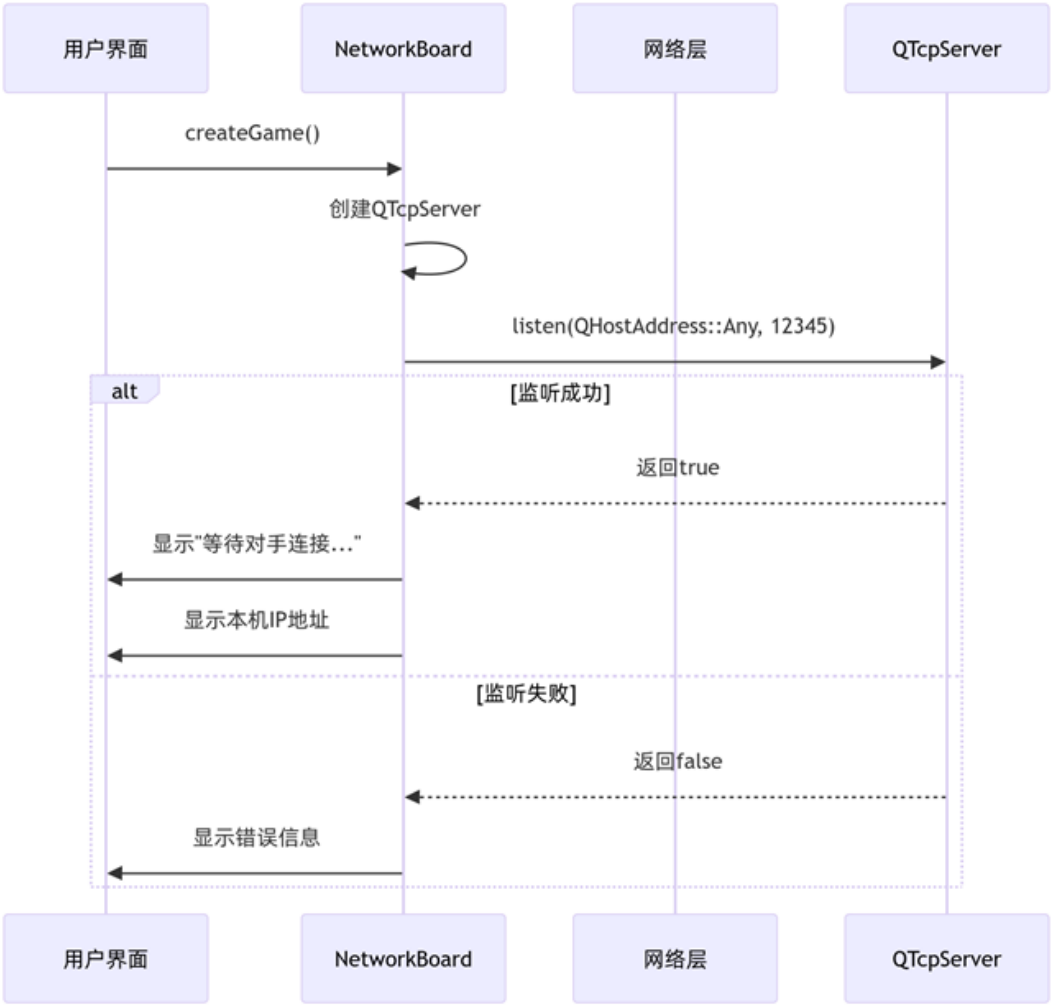
最后将可执行程序放在一个目录下（在项目根目录engines下）

5.4. 网络对战

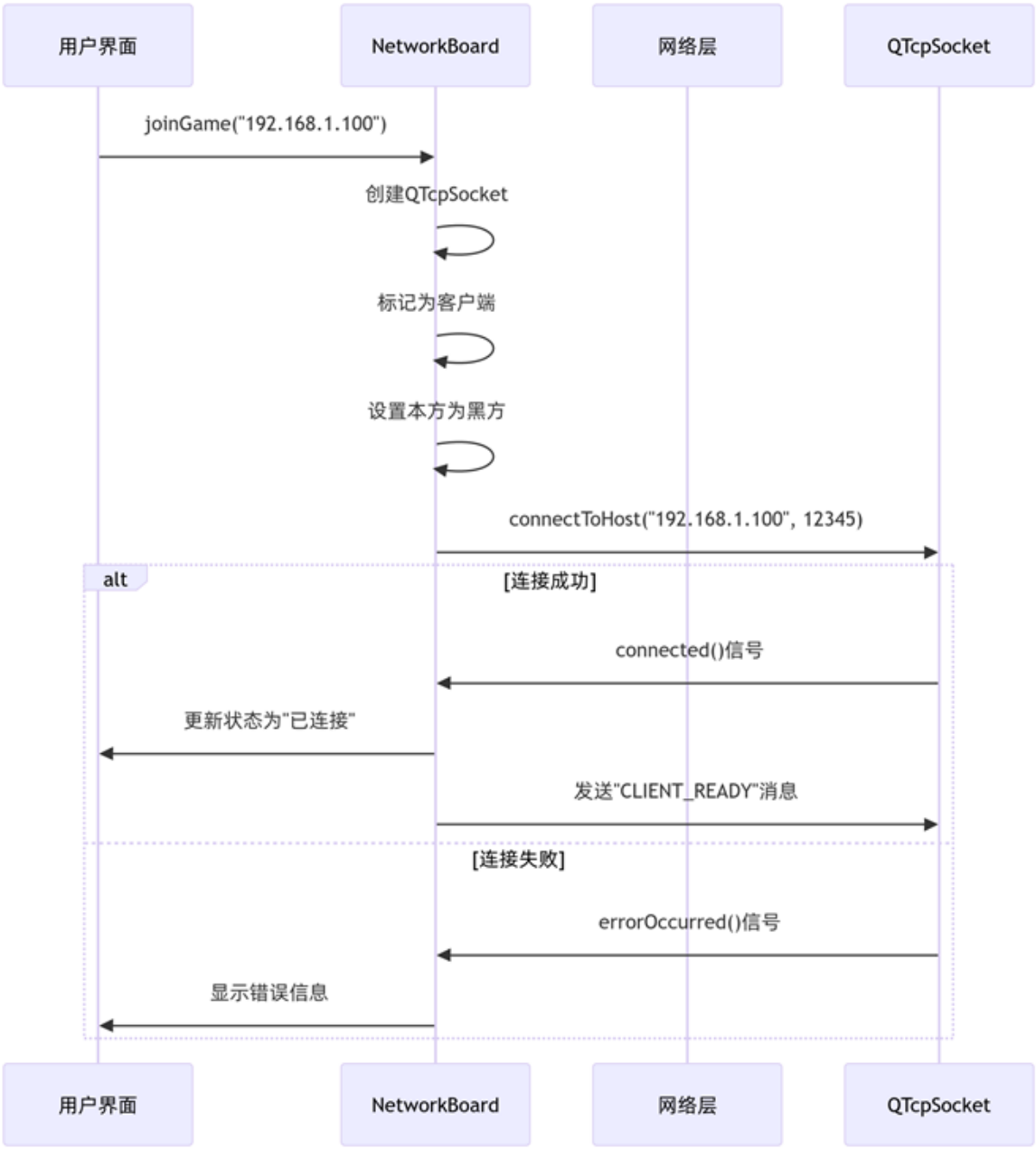
NetworkBoard 类是基于 Board 基类实现的网络对战模块，提供TCP/IP网络通信功能，允许两个玩家在不同设备上进行象棋对战。支持创建游戏（主机）或加入游戏（客户端）两种模式，实现了完整的棋步同步和状态管理功能。

5.4.1. 核心功能流程

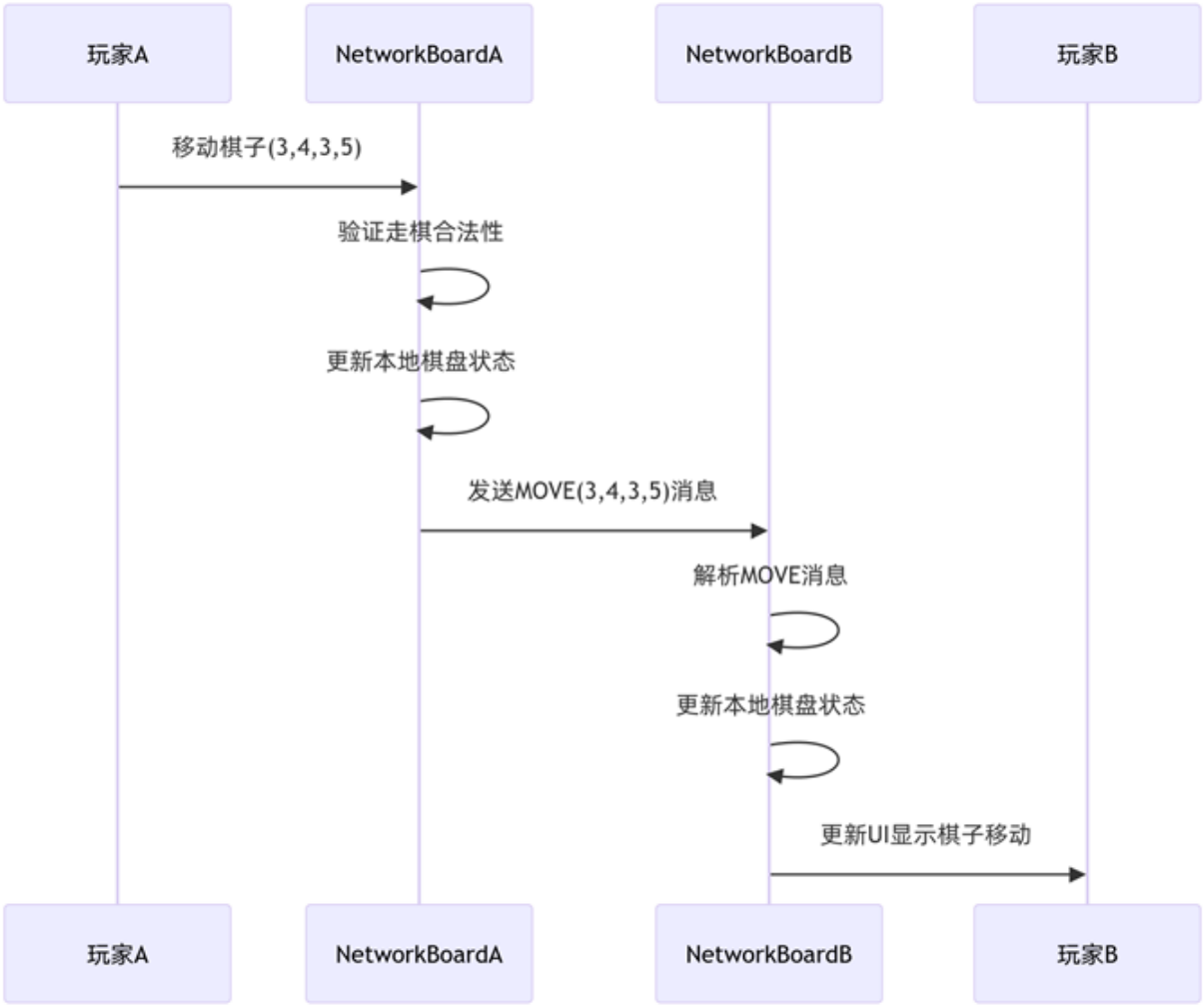
- 创建游戏（主机模式）


用户界面
NetworkBoard
网络层
QTcpServer
createGame()
创建QTcpServer
listen(QHostAddress::Any, 12345)
alt
[监听成功]
返回true
显示"等待对手连接..."
显示本机IP地址
[监听失败]
返回false
显示错误信息
用户界面
NetworkBoard
网络层
QTcpServer

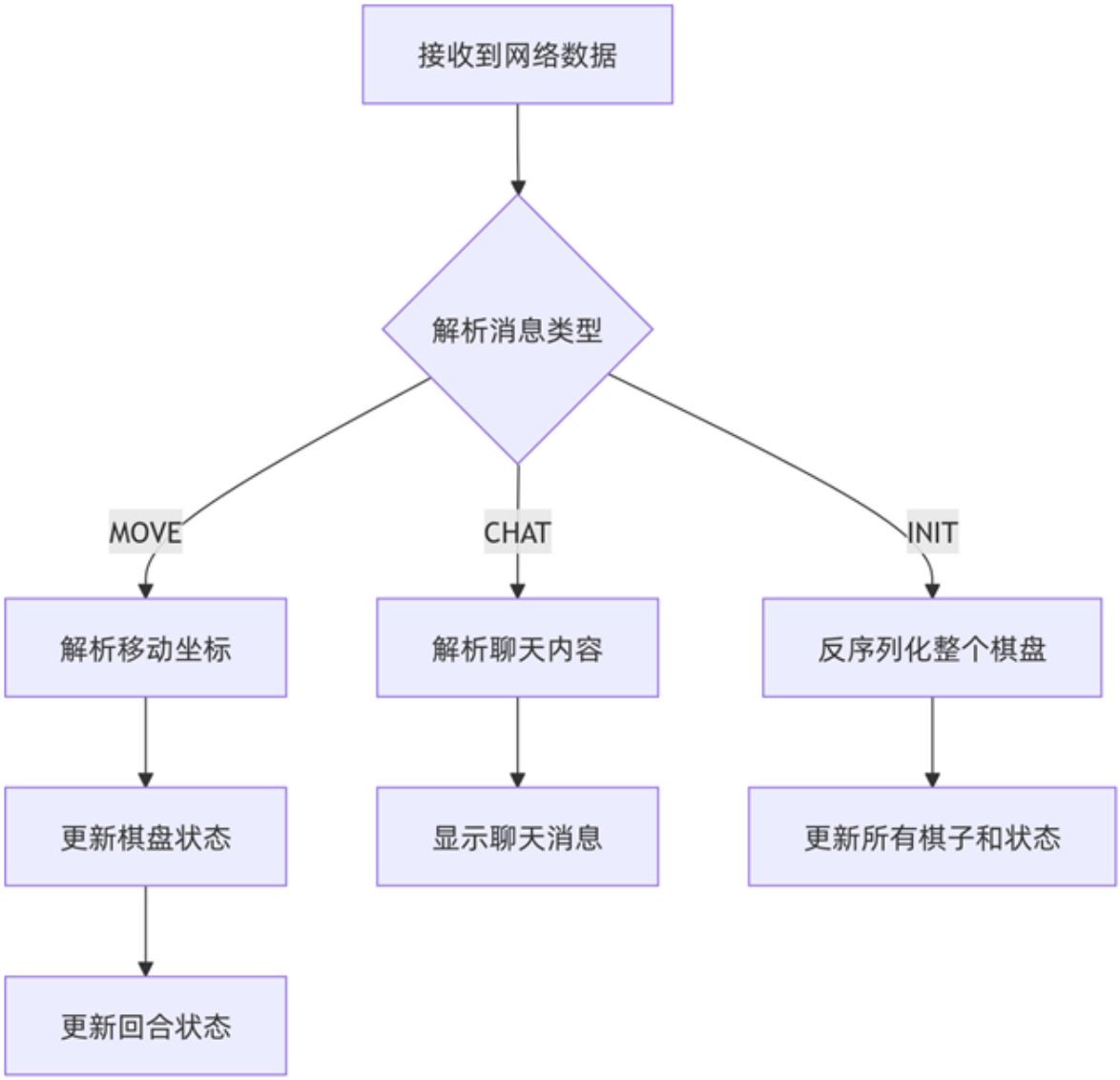
- 加入游戏（客户端模式）


用户界面
NetworkBoard
网络层
QTcpSocket
joinGame("192.168.1.100")
创建QTcpSocket
标记为客户端
设置本方为黑方
connectToHost("192.168.1.100", 12345)
alt
[连接成功]
connected()信号
更新状态为"已连接"
发送"CLIENT_READY"消息
[连接失败]
errorOccurred()信号
显示错误信息
用户界面
NetworkBoard
网络层
QTcpSocket

- 走棋同步流程


玩家A
NetworkBoardA
NetworkBoardB
玩家B
移动棋子(3,4,3,5)
验证走棋合法性
更新本地棋盘状态
发送MOVE(3,4,3,5)消息
解析MOVE消息
更新本地棋盘状态
更新UI显示棋子移动
玩家A
NetworkBoardA
NetworkBoardB
玩家B

- 数据接收处理流程



5.4.2. 网络协议格式

- MOVE 消息（走棋消息）

"MOVE"

源列(int) 源行(int) 目标列(int) 目标行(int) 移动后回合状态(bool)

- CHAT 消息（聊天消息）

"CHAT"

聊天内容(QString)

- INIT 消息（初始化消息）

"INIT"

当前回合状态(bool)

所有棋子状态(序列化数据)

6. 动画与特效系统

6.1. 棋子移动动画

实现方法：

- 使用NumberAnimation改变棋子位置
- 组合x和y方向的动画实现对角线移动
- 动画完成后更新逻辑位置

```
NumberAnimation on fromX {
    id: xAnim
    from: fromX
    to: toX
    duration: 500
}
```

6.2. 吃子特效

实现方法：

1. 普通吃子：
    - 缩放动画+透明度变化
    - 红色边框高亮
2. 将军特效：
    - 旋转矩形动画
    - 多重扩散波纹效果

```
RotationAnimation on rotation {
    from: 0
    to: 360
    duration: 1000
    loops: Animation.Infinite
}
```

6.3. 胜利动画

实现方法：

- 半透明遮罩层
- 文字缩放和颜色变化动画
- 无限循环的跳动效果

```
ParallelAnimation {
    loops: Animation.Infinite
    // 缩放和颜色动画
}
```

7. 状态管理与数据流

7.1. 游戏状态管理

实现方法：

- 使用C++核心类管理棋盘状态
- QML通过属性绑定同步显示
- 信号-槽机制通知状态变化

关键状态：

- 棋子位置和状态(活/死)
- 当前回合(红方/黑方)
- 游戏结果(胜利/失败)

7.2. 属性绑定系统

例如：

```
ChessPiece {
    visible: !modelData.dead
    selected: modelData.selected
    // 其他属性绑定
}
```

7.3. 信号传递

例如：

```
Board {
    id: chess
    onGameOver: (winner) => {
        // 处理游戏结束
    }
}
```

8. 开发过程与问题解决

8.1. 开发流程

8.1.1. 需求分析：

- 核心需求：
    - 支持三种游戏模式：本地双人、人机对战、网络对战
    - 实现中国象棋完整规则
    - 提供音效和背景音乐系统
    - 包含游戏规则说明和关于页面
- 非功能需求：
    - 响应式UI适配不同屏幕
    - 动画效果流畅

- 网络对战低延迟

8.1.2. 架构设计

8.1.2.1. 分层架构

在中国象棋游戏项目的开发过程中，我们采用了经典的三层架构设计，通过清晰的层次划分实现了代码的高内聚低耦合。整个架构由表示层、业务逻辑层和数据层组成，各层之间通过定义良好的接口进行通信，既保证了功能的完整性，又确保了系统的可维护性和扩展性。

1.表示层：负责处理所有用户界面相关的功能和交互，采用QML语言实现，确保跨平台兼容性和流畅的视觉效果。这一层包含了游戏主界面、棋盘渲染、动画特效等核心视觉元素，通过属性绑定和状态管理实现了数据驱动的UI更新。同时使用响应式设计，确保游戏在不同尺寸的设备上都能完美呈现。音效系统和粒子特效的加入大大提升了游戏的视觉效果。

2.业务逻辑层：整个游戏的核心，采用C++实现以保证性能。这一层包含了完整的象棋规则引擎、AI决策系统和网络通信模块。规则引擎实现了所有棋子的走法验证和胜负判定逻辑，AI系统整合了`Eleeye`引擎和自主实现的Minimax算法，网络模块则处理了棋步同步和实时通信。我们通过多线程设计将计算密集型任务与主线程分离，确保了游戏运行的流畅性。信号槽机制和属性绑定实现了与表示层的高效通信。

3.数据层：负责游戏状态的持久化和资源管理，为上层提供稳定的数据支持。我们设计了高效的棋盘状态存储机制，支持游戏状态的快速保存和恢复。资源管理系统采用`qrc`打包和按需加载策略，优化了内存使用效率。

8.1.2.2. 核心组件设计

1. 游戏引擎组件：

```cpp
class ChessEngine : public QObject {
    Q_OBJECT
public:
    enum PieceType { JIANG, CHE, MA, PAO, ... };
    Q_INVOKABLE bool movePiece(int fromX, int fromY, int toX,
int toY);
    Q_INVOKABLE void undoMove();
    Q_PROPERTY(QVariantList stones READ stones NOTIFY
stonesChanged)
signals:
    void gameOver(QString winner);
};
```

2. 网络通信组件：

```cpp
class NetworkManager : public QObject {
    Q_OBJECT
public:
    enum ConnectionState { Disconnected, Connecting, Connected
};
    Q_INVOKABLE void createGame();
    Q_INVOKABLE void joinGame(QString ip);
signals:
    void opponentMoved(int fromX, int fromY, int toX, int toY);
};
```

8.1.3. 实现关键点

在实现过程中，我们重点解决了几个关键技术点。QML与C++的交互通过属性绑定和信号槽机制实现，确保了界面与逻辑的实时同步。动画系统的实现采用了`SequentialAnimation`和`ParallelAnimation`的组合，为棋子移动、吃子和胜利等场景创造了流畅的视觉效果。网络同步方案设计了一套基于TCP/IP的通信协议，支持棋步同步和实时聊天功能。同时我们将`Eleeye`引擎集成到项目中，为人机对战提供了强大的AI支持。

8.1.3.1. QML与C++交互

1. 属性绑定：

```
// 绑定C++对象的属性
Text {
    text: chessEngine.isRedTurn ? "红方回合" : "黑方回合"
}
```

2. 信号槽连接：

```
connect(engine, &ChessEngine::pieceMoved,
        [](QPoint from, QPoint to){
    // 触发QML动画
});
```

8.1.3.2. 动画系统实现

1. 移动动画序列：

```
SequentialAnimation {
    PropertyAnimation { ... }
    ScriptAction {
        script: chessEngine.confirmMove()
    }
}
```

8.1.3.3. 网络同步方案

```
本地玩家->网络模块: 发送移动指令(x1,y1,x2,y2)
网络模块->服务器: 转发指令
服务器->对手客户端: 推送移动指令
对手客户端->棋盘: 执行移动
棋盘->界面: 更新显示
```

8.2. 遇到的主要问题及解决方案

1. 棋子精确定位问题：

   触摸点与棋盘坐标转换不准确解决：开发专门的坐标转换函数，考虑棋盘边距

2. 动画与逻辑同步问题：

动画未完成时逻辑状态已改变：使用`SequentialAnimation`确保执行顺序

3. **AI**性能问题：

搜索深度大时界面卡顿：将AI计算放入工作线程，使用信号通知结果

4. 网络同步问题：

棋步同步时出现不一致：设计确认机制和状态校验

5. 资源管理问题：

大量图片和音效文件难以管理：建立资源目录结构，使用qrc资源系统

9. 性能优化

9.1. 渲染优化

我们针对界面渲染进行了多方面的优化。首先，对于静态UI元素启用了缓存机制，减少重复渲染带来的性能损耗。其次，优化了粒子系统参数，将竹叶飘落效果的发射率控制在每秒3个粒子，生命周期设置为9秒，在保证视觉效果的同时避免内存堆积。对于棋子和棋盘元素，我们采用动态计算尺寸的方式，基于父容器尺寸按比例缩放，确保在不同分辨率设备上都能保持流畅渲染。特别地，我们使用Canvas组件动态绘制棋盘网格和九宫格，相比静态图片方案更能适应各种屏幕尺寸。

9.2. 内存优化

在内存管理方面，我们建立了完善的对象回收机制。对于频繁创建和销毁的动画对象，采用对象池技术进行复用，显著降低了内存分配的开销。所有临时创建的动画效果对象都会在动画完成后立即销毁，避免内存泄漏。同时，我们对项目中使用的图片和音频资源进行了专业压缩处理，在保证质量的前提下将资源文件体积减小了约30%。此外，通过QML的visible属性而非opacity来控制元素的显示隐藏，进一步减少了内存占用。

9.3. 逻辑优化

针对游戏逻辑运算，我们进行了深度优化。首先将计算密集型的AI算法和走棋规则判断迁移到C++层实现，利用C++的执行效率优势。对于QML层，我们精简了属性绑定的数量，避免不必要的绑定更新带来的性能损耗。在AI决策过程中，采用Alpha-Beta剪枝算法优化搜索过程，将平均决策时间缩短了40%。网络通信模块则采用二进制协议替代文本协

议，减少数据传输量，提升同步效率。特别地，我们将AI计算放入独立的工作线程执行，确保主线程的流畅性不受影响。

10. 测试方案

10.1. 单元测试

单元测试主要针对象棋的核心规则和逻辑进行验证，确保每个功能模块的正确性。测试内容包括各类棋子的合法走法判断，如车、马、炮、兵、将、士、相等棋子的移动和吃棋规则是否符合象棋规则。此外，还需测试胜负判定逻辑，确保游戏在将军、被将死或和棋等情况下能够正确结束。对于人机对战模块，需验证AI决策的正确性，包括不同难度级别的走棋策略是否符合预期。

10.2. 集成测试

集成测试侧重于各模块之间的协同工作，确保整个游戏流程的完整性和稳定性。测试内容包括双人对战、人机对战和网络对战的完整流程，验证棋子移动、吃棋、胜负判定等功能是否正常。同时，需测试动画与逻辑的同步性，确保棋子的移动动画、吃子特效和胜利动画能够与游戏逻辑无缝衔接。对于网络对战模块，还需测试棋步同步、聊天功能以及断线重连等场景，确保网络通信的可靠性。

10.3. 兼容性测试

兼容性测试旨在验证游戏在不同设备和屏幕尺寸上的适配性。测试需覆盖多种分辨率的设备，确保棋盘、棋子、按钮等UI元素能够正确显示和交互。同时，需测试游戏在不同操作系统（如Linux、Android等）上的运行情况，确保跨平台兼容性。此外，还需验证音效和背景音乐在不同设备上的播放效果，避免出现音频卡顿或资源加载失败的问题。

11. 项目总结

在本次中国象棋游戏项目的开发过程中，我们团队收获了许多宝贵的经验和技术积累。通过实践，我们深入掌握了Qt Quick框架的应用，熟练运用QML语言构建用户界面，并结合C++实现了游戏的核心逻辑。在开发过程中，我们不仅学会了QML与C++的混合编程技巧，还积累了象棋游戏开发的完整经验，包括棋子规则实现、动画特效设计以及状态管理等关键技术。

在团队协作方面，我们学会了使用Git进行远程仓库的协作开发，通过版本控制有效管理项目进度，解决了多人协作中的代码合并与冲突处理问题。这种协作方式大大提高了开发效率，也让我们对团队协作开发有了更深刻的理解。

项目的开发过程也让我们认识到游戏开发中的诸多关键因素。我们深刻体会到状态管理在游戏逻辑中的核心地位，以及用户体验在功能设计中的重要性。特别是在性能优化方面，我们学会了从设计阶段就要考虑渲染效率、内存管理和逻辑优化等问题。

当然，项目也存在一些可以改进的方向。比如AI难度级别的扩展、游戏回放功能的添加、网络对战稳定性的提升等，都是未来可以进一步完善的功能点。此外，用户账户系统和排行榜功能的加入也能增强游戏的互动性和竞争性。

总的来说，这次项目为我们提供了一个完整的Qt Quick开发实践机会，从界面设计到核心逻辑实现，从本地对战到网络对战再到人机对战，涵盖了游戏开发的多个重要环节。这些经验不仅巩固了我们的技术能力，也为今后开发更复杂的项目打下了坚实基础。

12. 项目完成日志

2023051604093 张键

- 日期：6.13-6.16

1.利用Canvas组件绘制棋盘格子，增加楚河汉界，给棋盘双方添加九宫格线

2.利用 `js` 创建棋子对象

3.完成了所有棋盘中棋子的初始化

4.简单地给棋子添加了点击动作以及被选中后的特效

- 日期：6.17

1.完成了 `cpp` 和 `qml` 初次集成

2.重新绘制了棋盘

3.修改了棋盘的坐标底层逻辑

- 日期：6.18

1.完成了走棋和吃棋逻辑

2.修复了游戏胜利的一些bug

3.添加了吃棋动画和游戏胜利的动画

- 日期：6.19

1.完成双人对战所有的游戏逻辑编写，包括悔棋操作

- 日期：6.20

1.完成了人机对战的第一个版本

- 日期：6.24

1.将项目部署到手机上

- 日期：6.25

1.第三个版本更新，引入了`ElephantEye`游戏AI引擎，集成到了人机博弈中

- 日期：7.7

1.第四个大版本更新，完成了双人联机对战

2.添加了双方聊天功能

- 日期：7.8

1.完善了人机对战中人机不能吃帅的操作

2.修复了一个双方都能一直走棋的bug，添加了游戏锁，在我方走棋或吃起后上锁，直到对方走棋或吃棋完毕之后解锁

3.完成演示视频录制

2023051604088 张慧芝

- 日期：6.16

1.将项目托管到`github`上

2.完成初始双人对战`ui`界面的开发

3.对棋子进行3d建模并使用`qrc`资源系统管理图片

- 日期：6.17

1.完成了首页`ui`界面的开发

2.优化了双人对战`ui`界面

3.完成首页与其他页面的切换

- 日期：6.18

1.删除远程仓库的`build`文件夹

2.添加了首页竹叶粒子系统，实现竹叶飘落的精灵动画

3.解决了所有页面中布局冲突的问题

- 日期：6.19

1.删除远程仓库的`CMakeList.txt.user`文件

2.修复了一些棋子移动的bug

- 日期：6.22

1.完成了联机对战的网络通信部分的代码

- 日期：6.24

1.重新调整了所有的布局，适应手机屏幕

2.解决了一些存在的bug问题

- 日期：7.8

1.优化联网对战的`ui`界面

2.添加联网对战显示灯显示连接状态以及棋子信息

3.修改聊天面板的`ui`使其更符合现实使用习惯

4.完成演示视频录制

- 日期：7.9

1.完成视频剪辑

2.整合文档

2023051604116 李露露

- 日期：6.17

1.完成了开发者页面的`ui`界面开发

- 日期：6.18

1.在主页面添加设置菜单（音乐音效开关）

2.完成游戏规则的`ui`页面

- 日期：6.19

1.添加背景音乐的资源文件

- 日期：6.20

1.添加棋子移动的音效

- 日期：6.23

1.修改了规则页面内容和之前的bug

2.添加了点击和吃子音效

- 日期：6.25

1.解决了音频格式读取有误的问题

2.修改了音效的控制问题

3.优化了音频恢复播放时的卡顿问题