# 需求分析

## 场景和用户需求

1. 玩家发起一个新游戏
   1. 玩家登录系统后，点击“发起一个新游戏”，等待其他玩家的加入
   2. 系统提示给用户一个提示信息：“正在等待其他玩家加入”
2. 玩家加入一个游戏
   1. 玩家登录后，可以浏览所有“等待其他玩家加入”的游戏
   2. 从列表里面选择一个“等待其他玩家加入”的游戏，并点击“加入”
3. 玩家认输
   1. 玩家可以主动认输，表示对方赢得比赛，得1分，自己得0分
   2. 认输后可以选择是否继续
4. 玩家和棋
   1. 在两个玩家的棋子都没有被吃光的情况下，一个玩家发起和棋的请求
   2. 另一个玩家可以选择是否接受和棋
      1. 如果接受，则结束本局，双方都记0分
      2. 如果不接受，比赛继续，直到一方认输或棋子被吃光
5. 玩家退出
   1. 如果一个玩家由于“和棋”请求被拒，或其他原因，想要退出游戏，系统可以结束该游戏
   2. 双方玩家将本局游戏的分数累加到历史分数
6. 玩家的棋子被吃光
   1. 玩家的棋子被吃光以后，游戏结束，对方得1分，自己得0分
   2. 系统提示玩家是否继续下一局
      1. 如果双方玩家都同意继续下一局，重新开始比赛
      2. 如果有一方不同意，则退出比赛，双方玩家将本局游戏的分数累加到历史分数
7. 玩家下棋过程
   1. 按照规则走子（翻子、移动、兑子、吃子）

## 产品对象分析

根据上述的用户场景分析，我们可以抽象出如下几种对象

1. 玩家（Player）
   * 属性：名字、当前所在棋盘、当前游戏分数、历史分数
2. 棋盘游戏管理者（ChessboardManager）
   * 维护所有处于即将开始/已开始的棋盘
3. 棋盘（Chessboard）
   * 棋盘维护一个4\*8的二维数组，元素或者为null，或者为某个棋子
   * 状态：standby（游戏刚刚开始，处于洗牌阶段）、started（洗牌结束、游戏开始）、ended（游戏结束）
4. 棋子（ChessPiece）
   * 每个棋子都有自己的名字、权重、颜色、攻击方式（隔子攻击/相邻攻击）等属性

### 关于棋子的权重

棋子的权重值可以用int值来表示

|  |  |
| --- | --- |
| **棋子类型** | **权重** |
| 将帅 | 6 |
| 士 | 5 |
| 象 | 4 |
| 马 | 3 |
| 车 | 2 |
| 炮 | 1 |
| 兵卒 | 0 |

#### 查看A棋子是否可以吃到B棋子

* 先得到A和B的权重的差，
* 如果差小于6并且大于0，则说明A可以吃掉B
  + 将帅不可以吃掉兵卒
* 如果差等于-6，说明A也可以吃掉B
  + 兵卒可以吃掉将帅

#### 查看A棋子是否可以兑掉B棋子

* 先得到A和B的权重的差
* 如果差等于0，则表示A可以兑掉B

思路：可以在棋子的对象上加一个方法attack，即A.attack(B)，返回值？

## 下棋过程分析

当两个玩家都加入一个游戏以后，会经历如下几个步骤完成下棋操作

### 棋盘混洗

* 把4\*8个棋子的每一个棋子，随机分配到棋盘的二维数组上
* 随机选取一个玩家，授予他“走子”的权利

### 玩家走棋

玩家走棋的过程，以UI驱动的方式来进行讨论

#### 翻开或移动

1. Player点击一个棋子
   1. 如果该棋子的状态为“未翻开”，则执行“翻开”操作
      1. 如果当前Player的棋子颜色尚未设定，则设定该Player的棋子颜色为当前被翻开棋子的颜色
      2. 同时另一个Player的颜色被设定为另一个颜色
   2. 如果该棋子的状态为“已翻开”并且不是自己棋子的颜色，则提示该Player非法操作，让他重新点击其他的棋子
   3. 如果该棋子的状态为“已翻开”并且是自己棋子的颜色，则“选定”该棋子，等待下一步执行的操作
      1. 如果该Player继续点击该“选定”的棋子，则表明要取消操作，Player可以继续点击任何一个子
      2. 如果该Player点击了“选定”棋子的相邻位置，并且相邻位置为空，则“移动”该“选定”棋子到当前位置
      3. 如果该Player点击了“选定”棋子的相邻位置，并且相邻位置有子，则根据具体情况执行“兑子”或“吃子”的操作

#### 兑子或相邻吃子

前提：

* 已经有一个“选定”的己方的棋子，
* 该“选定”棋子的相邻棋子是对方的棋子，
* 并且对方棋子是“已翻开”的状态

可以进行的操作

* 根据“选定”棋子与对方棋子的权重，决定是兑子、吃子还是什么都不能做
* 如果是兑子，则
  + 把这两个子同时从棋盘中删除
  + 分别从两个Player中删掉对应的棋子
* 如果是吃子，并且该“选定”的棋子不是“炮”，
  + 把对方棋子从棋盘删除
  + 把对方棋子从对方Player中删除
* 如果什么都不能做，则
  + 提示Player，这是一个非法操作，继续保持己方棋子的选定状态，并等待该Player的下一步操作

#### 炮的吃子

前提：

* 已经有一个“选定”的己方的棋子，并且该棋子是“炮”

可以进行的操作

* Player点击一个与当前“炮”隔着一个子的对方棋子（已翻开），
  + 把对方棋子从棋盘删除，空出位置
  + 把己方的“炮”放到刚刚删除的对方棋子的位置
* Player点击相邻位置
  + 如果相邻位置为空，则移动“炮”到相邻位置
  + 如果相邻位置不为空，则提示Player非法操作，保持“炮”的“选定”状态，并等待该Player的下一步操作

## 底层架构分析

使用B/S模式构建系统，前后端采用WebSocket进行通信。所有的数据都维护在后端，前段订阅后端的数据更新。当后端的数据变化时，前段会立即收到通知并对页面做相应的渲染。

Player1

Player2

棋盘游戏

前端

后端