Part 3

3.1 实现顺序及思路

考虑使用 canvas 对网页进行开发。

思路大体上是先写一个能够画出六边形的代码，然后通过循环绘制整个棋盘。

在画棋子部分,我先尝试将所有棋子的实际坐标进行保存,`{x: x, y: y, color: color}`但是后面发现这么写对后续开发影响较大，不适合用来检测胜负，也不方便维护。故采用二维数组`chesslist[x][y] = color`，其中 color 为颜色在 colorlist 中的索引值。

输出的内容完成后，接着完成输入的部分。通过为 canvas 绑定事件，使得在点击时画布能够做出响应，并通过计算离鼠标最近的六边形块坐标来判断用户下棋的位置，更新棋局信息。这样还能保证用户在同一位置只能下一次棋，不会出现显示上的问题。接着我实现了 Hex 棋独特的先手交换原则。

此时大部分功能已经实现。

在下棋的时候不止更新棋局，也更新下棋记录，就可以实现棋局的撤回与重做；添加一个表单用于更新棋子的颜色等信息，就可以实现棋盘样式的修改。

原先的棋盘没有边框，参考给出的例程，我通过画三角形的方式为六边形棋盘填充边框，用来提示玩家胜负条件。

再综合修改，完成整体网页的实现。

3.2 代码细节

`hex.js`是整个函数的核心部分，在文件的前一部分，我先定义了程序所需要的变量，其中`hexCenters`用于维护所有六边形棋盘在实际画布中的坐标值；`chesslist`代表棋盘上每个棋子的颜色；`current_color`代表当前下棋的颜色，`first_chess`用于维护“先手交换原则”，`record`，`redu`用于维护撤回与重做，这是两个栈，在撤回时`record`中的文件出栈，进入`redo`中，重做时`redu`的元素重新进入`record`中。

函数模块，我设计了画三角形，圆形（棋子）和六边形（棋盘）的函数。

`drawBoard`是程序的核心函数，他负责更新棋盘状态，棋子状态，绘制棋盘边框。

`getHexRowCol`，`getHexCoordinate`负责转换实际坐标点与棋盘中的坐标。

在这个程序中，我通过检测离鼠标点按坐标最近的六边形坐标来判断选择下棋的位置，并更新 chesslist, record 使下棋生效。

3.3 新增功能

初步实现功能之后，我为棋盘增加了以下功能。

1. 棋盘样式更改

由于棋盘颜色`boardColor`, 维护两个玩家颜色都存在变量中，于是考虑设计一个表单，通过监听按钮是否被按下来更新棋盘的颜色样式。大小样式也存放在了 size 变量中，但是由于画布无法自适应改变大小，棋盘坐标较难设定，于是放弃了可自定义棋盘大小的功能。

2. 撤回与重做

   通过两个栈互相出栈入栈来实现棋面的复原，在`record`数组中存放整个棋局的信息，确保在撤回后仍然可以使用 “先手交换”。

3. 胜负判定

   已经存有棋局数据，通过 BFS 算法较容易可以实现胜负的判定。例如玩家一需要使得上下联通，则将最上边的所有棋子加入队列，在出队的同时将与他相连且未遍历过的同色棋子加入队列，如果能用这种方式遍历到另一边，则该玩家胜利。

4. 棋局信息显示

   通过格式化`record`数组中的内容，并将其添加至`moveHistory`文本框中，就可以实现棋局信息显示。这个文本框中的内容只在下棋或撤回重做时更新。

5. 规则显示

   新增一个 button，按下时通过 alert 来显示 Hex 棋的规则。