draughts_online

实现功能：

1. 实现网络对战功能。

- 显示 IP，能自定义端口号并进行监听；
- 能够填写对方 IP 及端口号并进行连接；
- 能够实时对对方的走棋在棋盘上做出响应，同时本方的走棋对方也能看到。

2. 实现完整功能的国际跳棋。
3. 认输功能及求和功能。
4. 音效。

运行效果：

1. 能对格子选中，高亮，标记，显示是否错误

2. 能够高效而正确地求解

模块划分：

主逻辑 (./logic 文件夹下)

类	功能
LogicController	控制游戏主逻辑，诸如重玩，撤销与回撤，正确性检查，发送信号更新界面

界面 (./UI 文件夹下)

类	功能
MainWindow	用于选择关卡
GameBoard	游戏界面
SudokuGrid	数独格子组件
Inputboard	输入数字框组件
GridBtn	继承QLabel而写成的按钮控件

求解器 （./solver 文件夹下）

类	功能

Solver	Algorithm X 的抽象算法，实现为一个函数对象，通过调用DancingLinks接口描述
DancingLinks	实现dancinglinks基本操作，数据底层操作调用DLNodesContainer的实现
DLNodesContainer	链表节点的容器，规定了DancingLinks操作所需的底层基本操作，继承重写可以实现不同种类数独
StdSudokuNodesContainer	用于表示9 x 9标准数独的DLNodesContainer

数独加载 (./loader 文件夹下)

类	功能
Loader	所有加载器的基类，暴露唯一对外界有用的接口Loader::load()方法
FileLoader	通过磁盘文件资源进行加载，实现功能中的关卡设计
RandomGenerator	随机生成数独游戏

测试 (./test 文件夹下)

出于熟练度考虑没有在本次开发中使用测试框架，而是自己通过编写一个test.cpp在其中利用宏定义+预编译的方法测试各个模块，让我最后一天开发效率明显提升。

关于求解与生成算法

1. 求解算法使用dancinglinks数据结构，使用 Algorithm X 进行求解
2. 我将 Algorithm X 单独抽取出来作为一个框架写在Solver中，通过调用数据结构DancingLinks的接口进行求解，DancingLinks 数据结构用于隔离底层操作，调用封装好的DLNodesContainer进行操作，通过实现继承重写DLNodesContainer 的少量最基本底层操作可以实现对求解问题的扩展
3. 生成算法使用知乎用户提供的一种方法：(1) 随机生成11个位置，调用solver计算一个解答形成终盘。(2) 之后随机挖洞，每挖一个洞调用 Algorithm X 判断解答是否唯一，如果不唯一就回溯，尝试挖洞次数越多，统计上获得空格多的数独可能性就越大
4. 随机生成难度控制：可以通过生成算法步骤(2)中的随机尝试次数进行控制，实验发现尝试次数30次左右可以产生简单难度的数独，尝试次数达到接近100次左右数独的空格数可以达到50个左右，效果相当显著

大作业感想与心路历程

1. 耗费了最大精力的就是类和模块的接口设计以及Solver的实现，对Solver最后实现了算法与数据表示的解耦让我感到很高兴。
2. 时间安排还是不太合理，前松后紧，最后一天的时候尽管平均两小时调过一个功能，写了一整天，仍然在检查前没有全部完成。。。
3. 充分体会到了写测试的好处，编写测试尽管要写更多代码但是会显著加快开发进度。

4. 一点遗憾：本来想实现一些数独变种（比如对角数独），dancinglinks的数据表示部分解耦就是为了这个目的，但是时间对于我来说确实太紧张，而且最后前端界面和逻辑部分有一些hardcode了，导致没有能够完成。

5. 关于undo和redo的实现：本来以为这两个功能实现起来挺麻烦的，但是后来发现因为数据部分完全存储在logic模块，其他部分只要logic模块的数据定下就确定下来了，所以最后采用保存与恢复logic模块的状态来实现。最终undo和redo功能总共就用了大概一个小时，各用了20行左右逻辑很简单的代码就实现了，这个模块的快速扩展是这次开发中的一个惊喜。

Acknowledgement:

图标全部来自：https://www.flaticon.com/ 再此向图标作者表示感谢！