QSnake 贪吃蛇设计文档

项目结构

项目源代码按文件夹组织成如下形式：

- core
- game
- log
- res
- ui

`core`文件夹存储游戏的核心组件。包含`Controller`，为游戏的核心控制系统。
`game`文件夹下存储游戏中需要用到的各个类，例如`Snake`贪吃蛇类，`Food`食物类等。
`log`文件夹下存储一个自定义日志类。
`res`文件夹下存储资源文件。
`ui`文件夹下存储游戏的显示组件。包括`MainWindow`主界面，`GameWidget`游戏窗口等。

设计概述

游戏元素

游戏元素如传统的贪吃蛇游戏那样，包括贪吃蛇本蛇`Snake`，蛇的食物`Food`，以及蛇讨厌的砖块`Brick`。这些元素都存储在`game`目录下。
游戏的图像显示是像素化、网格化的。所有的元素会被渲染到一块方形的「草坪」上。所以，存储各个元素的位置采用的是一个具有两个坐标的点。
坐标系统以屏幕的左上角为原点，x轴沿横向，y轴沿纵向。各个元素的位置就以在此坐标系统中的坐标存储。
在这个游戏中，贪吃蛇的形状是规整的，为了尽可能节约内存占用，贪吃蛇的存储实现方案时，仅存储贪吃蛇各个顶点的坐标，保存为一个线性表。渲染蛇时，只需要顺次连接各个顶点即可。具有相同规整形状的砖以同样的方式进行存储。食物仅仅为一个点，只需要简单地存储一个点对象。
蛇仅仅吃普普通通的苹果也是会吃腻的，经过人工干预，食物还有不同的属性，例如冰冻的、火烤的等等。冰冻的苹果会被渲染为冰蓝色，能够让蛇跑得慢一些，喘喘气；火烤的苹果则是红色的，能够让蛇跑得更快。
墙和砖块是令蛇讨厌的存在，蛇也不知道为什么它如果撞到了墙或砖块就会死。更奇妙的是，砖块还会时不时变换位置。

游戏功能实现

从总体上讲，程序可分为三个部分，`game`组件负责实现各游戏元素的逻辑，`ui`组件负责生成用户界面，`core/Controller`负责`game`与`ui`之间的交互。三者在一定程度上相互隔离，期望实现代码良好的可扩展性。
程序中负责游戏核心控制的组件为`Controller`类。
`Controller`负责管理游戏中的各个元素，统领并协调各元素之间的数据和逻辑交互。
当用户进入游戏界面时，一个`Controller`对象会被创建。各个游戏元素是Controller的一个对象，也会在此时被初始化。
Controller中设计了`updateGame`函数，对游戏进行更新，每更新一次，蛇移动一步，同时进行碰撞检测。
碰撞检测通过判断蛇的坐标是否存在与食物、墙、砖块以及自身等元素的坐标重合来检测是否有碰撞。

由于前面提及的存储实现，该过程是比较高效的。
`generate` 函数生成一个位置随机的食物。
`restart` 函数重置各个成员和游戏元素，用于重新开局。
Controller中其它一部分函数实现了`ui`与`game`间的数据传递。一系列getter和setter抽象的数据获取与赋值。
除了Controller之外，游戏中还应用了一些信号槽，通过释放信号来引起另一个函数的响应。

用户界面

游戏的主界面是`MainWindow`。`MainWindow`内用一个`WidgetStack`存储需在`MainWindow`上显示的子窗口，即首页（欢迎界面）`WelcomeWindow`，游戏窗口`GameWidget`。
运用信号槽机制，连接`WelcomeWindow`中开始按钮的点击与`MainWindow`中进入对应游戏模式的行为。
进入游戏后，用户通过操纵方向键改变蛇的移动方向，双人模式下用 W, A, S, D 操纵第二条蛇。连按或长按 E 键可以实现加速。

AI的实现

本游戏中的贪吃蛇AI使用广度优先搜索(BFS)实现。
首先，对整张地图除蛇身和砖块之外的格子进行扫描，计算每个点到食物的哈密顿距离。具体做法是：

1. 将食物所在点标记为0。将其入队
2. 队头元素出队，将其上下左右四个点入队，并将它们的值标记为队头元素的值+1
3. 重复2过程，直至每一个点都被标记这样我们得到了一个矩阵，对于蛇而言，只要顺着值减小的路径走，就能爬到食物处。

代码风格

代码风格参考了Google C++ Style Guide。所有类成员变量统一用小写字母和下划线连接，所有成员函数统一使用驼峰命名法。
单行代码长度与字符间空格、缩进等都做了统一。