小学期第一周 Qt 大作业——贪吃蛇

计 96 凌壮 2019011339

1. 引言

本次作业的要求是利用 Qt 独立实现经典游戏贪吃蛇，并且增加了存档读档的功能。在完成作业的过程中，我发现实现游戏内在逻辑并不算难，工作量主要集中在**搜索并学习 Qt 独有的接口和处理因不熟悉 Qt 写出的 bug** 上，然而一旦逐渐掌握了 Qt 的特性，开发效率便大大提升。总体来说，这次作业对我来说是难度适中的挑战和很好的锻炼。

2. 成果

图 1. 效果图

作业文档中要求的功能均已实现(包括贪吃蛇基本操作、障碍设置、存档与读档)，与具体要求不同的是，程序中**用“Reset” + “Start”代替了“Restart”**，而且**在暂停状态下可以按下“Start”继续游戏**。除此之外，还添加了**调整贪吃蛇移动速度**的功能并且对贪吃蛇和游戏界面做了美化。

3. 实现

3.1 后端

3.1.1 游戏主体逻辑

创建一个**一维数组**，将下标映射为游戏画面中的坐标，数组中储存不同的**值**(0~4)分别对应画面中的**空地**、**蛇身**、**果实**、**障碍**、**蛇头**。游戏过程中的事件会**改变数组中储存的值**，而前端的函数在每一帧更新时会根据**值**的改变**绘制**相应的图形，由此便可以实现蛇的移动等效果。

3.1.2 初始化与状态记录

将**游戏状态**、**贪吃蛇移动的步数**、**目前获得的分数**、**贪吃蛇前进方向**、**单帧间隔(也即贪吃蛇移动速度)**、**是否存在果实**、**进食量**、**贪吃蛇各节坐标(存入 QList)**设为私有变量储存，并且在每次调用初始化函数 init()时重置。

游戏状态共有四个，分别为**初始状态**、**运行**、**暂停**和**结束**。不同的游戏状态会触发不同的相应，用户也只能在**特定**状态下进行某些操作，所以状态记录很有必要。

3.1.3 时间监听

这是游戏得以运行的核心，创建一个 Qtimer 类的对象，每隔设定好的时间便会发出 timeout()信号，作为槽函数的 timeToGo()便可以处理游戏逻辑，在**游戏状态**为**运行**时使游戏一帧一帧地进行。

3.1.4 蛇的移动及障碍、自噬检查

首先，通过 Qt 内置的 keyPressEvent()函数监听键盘的输入，上下左右键可以相应的改变**贪吃蛇前进方向**(不可直接转向反方向)，在每一帧内，根据方向将需要的点 push 入**贪吃蛇各节坐标**的头部，并将尾部的点 pop 掉，便实现了贪吃蛇的移动。而在移动后，重绘前，将会判断**蛇头**对应坐标的**值**是否为**蛇身**或**障碍**，若是，则将**游戏状态**改为**结束**，游戏结束，弹出游戏结束提示框。

3.1.5 果实刷新与蛇身增长

利用 qsrand()和 qrand()函数生成**数组**下标范围内的**随机数**，并检测**随机数**对应的**值**是否为**空地**，若是，则将**值**改为**果实**。并将**是否存在果实**设置为 true。若存在果实，则不再生成果实。

若**蛇头**对应坐标的坐标为**果实**，则将**进食量**设置为 3，当**进食量**大于 0 时，便不再 pop 尾部的点，转而将**进食量**减 1。由此便可以实现蛇身增长 3 格。

3.1.6 障碍设置与取消

首先，将 setMouseTracking 设置为 true，再利用 Qt 中的 mouseMoveEvent()和 mousePressEvent()函数追踪鼠标的位置以及监听鼠标按下事件。当鼠标在**游戏区**内进入了某个网格时，将鼠标的位置映射成**数组**的下标，前端的函数会将对应的网格绘制成“**预障碍**”颜色(灰色)；鼠标按下时，若该点为**空地**，则将对应点的**值**改为**障碍**，反之若为**障碍**，则改为**空地**。

3.1.7 存档与读档

需要**存档**时，利用 QFileDialog::getSaveFileName 调用系统的文件保存窗口，并保留用户选择的文件路径。之后用 Qt 针对 json 文件格式设计的各种接口储存**数组**、**游戏状态**等数据成员(此处给 QFile 对象的权限是 WriteOnly)。

读档时，利用 QFileDialog::getOpenFileName 调用系统的文件打开窗口，同样用 json 的接口加载需要的全部数据成员。

3.1.8 状态交互

用户在特定**游戏状态**下能进行的操作是受到限制的，此逻辑的实现依靠 Qt 控件内置的 setEnabled()函数。在时间响应槽函数 timeToGo()中，将根据**游戏状态**按作业文档中的要求设置控件的可触发性。

3.2 前端

3.2.2 布局

利用 Qt 的 UI 设计功能可以很方便地完成窗口的布局，只需要将特定的的组件拖曳至指定的区域，再调整大小、相对位置等属性即可。

在此实现中，游戏**主窗口**的大小被固定为 1550 * 1250，窗口顶部设有**菜单栏**，在下拉菜单中有 6 个可触发的控件。窗口主体部分为**游戏区**，游戏画面的呈现即在此处。在**游戏区**右侧分布有**游戏运行时间**、**得分**的记录，6 个功能各异的按钮以及**调速杆**。在**主窗口**的底部，设有位置固定的**工具栏**(位置固定是为了避免用户拖动影响窗口的整体布局)，其上也有 6 个控件，对应着作业文档中要求的功能。

3.2.1 图形绘制

Qt 中的 paintEvent()函数和 QPainter 对象可以满足绘图的需求。遍历整个**数组**，根据不同的**值**在窗口中对应的像素区域绘制不同颜色的的矩形。且在每一**帧**都调用 update()函数实时更新游戏画面，便可以让用户体验到“动态感”。

3.2.3 控件的信号与槽

Qt 的 UI 界面中，有一个很实用的“**转到槽**”功能，选中一个控件便可以直接生成某种触发状态下的槽函数，极大地提高了开发效率。鉴于**模块化编程**思想，此实现中将不同**游戏状态**下应执行的函数独立出来，在不同控件的槽函数中只需要调用相应的独立函数即可。此外，**菜单栏**和**工具栏**中的控件为相同的 QAction 对象，也减少了代码的重复。

图 2. 转到槽

4. 高光

4.1 美化

游戏画面中，各个矩形的区别仅仅只体现在颜色上未免过于单调，尤其是**蛇身**与**蛇头**，在**暂停**状态下甚至无法区分**贪吃蛇前进方向**。在此实现中，对**蛇头**进行了重绘，添加了眼睛和蛇信子，并可以朝向不同的方向，使用户有更好的感官体验。此外，还给主窗口与信息提示窗口添加了 Icon。

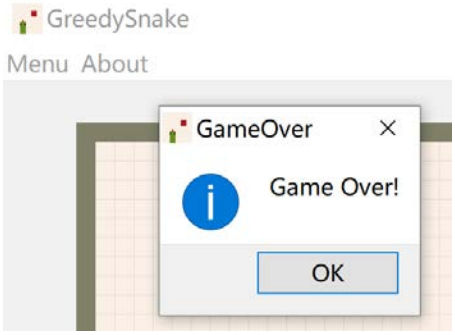

图 3. Icon

4.2 Json

在数据存储方面，选择了 json 这一跨平台的文件格式，利用 QJsonObject 和 QJsonArray 可以高效地进行数据的存储与读取。

4.3 Speed

这是此实现中额外增添的功能，在 UI 中设置一个 Horizontal Slider，通过 valueChanged()槽函数将它的**值**与数据成员**单帧间隔**做成一一映射，便可以实现贪吃蛇的速度调整。

5. 错误与处理

5.1 Focus

在 UI 中添加了按钮之后，发现上下左右键再也无法对贪吃蛇进行控制，上网查阅后得知要将按钮的 focusPolicy 从 strongFocus 改为 noFocus。这样方向键便不会再默认聚焦在按键上。

5.2 连续吃到果实

在游戏测试过程中，触发了在蛇身增长阶段再次吃到**果实**的小概率事件，结果因为后端逻辑错误，蛇身只能增加 3 格而不是 6 格，将相关逻辑从“设置为 3”改为“增加 3”便修复了 bug。

5.3 连续操作引发的自噬

在测试过程中还发现，如果用户在一帧内连续操作两次或以上，就有可能导致**贪吃蛇前进方向**直接改为反方向，从而引发游戏结束判定。解决方案是增设一个**操作队列**，

用户的键盘输入不再直接改变**贪吃蛇前进方向**，而是记录在**队列**中。每一帧内，若**队列**不为空，则从**队列**头部 pop 出一个**值**作为**贪吃蛇前进方向**，否则沿用之前的方向。

6. 总结

完成本次大作业让我获得了极大的成就感，也提高了搜索、学习、构建小工程的能力，我会将本次作业的经验总结完善，以期在今后的编程实践中运用自如。