Battle City——第三次课程设计报告

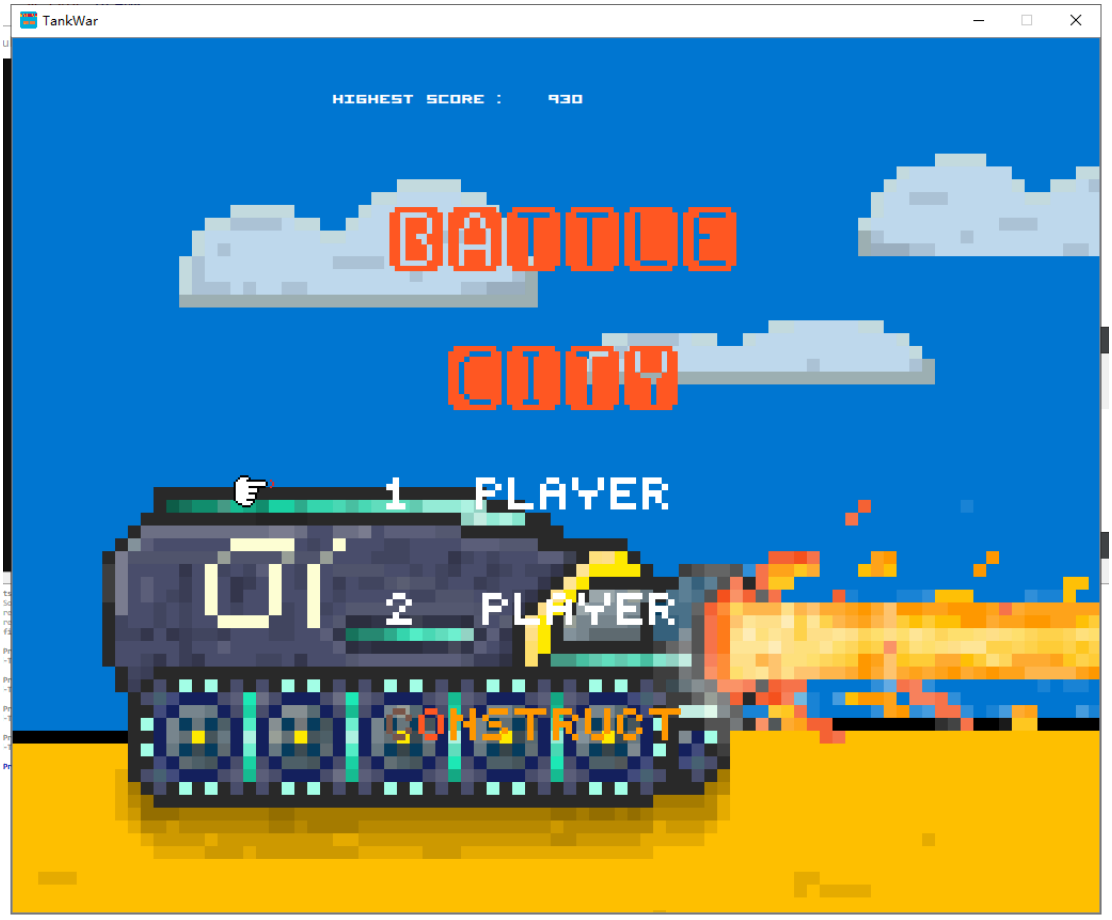
姓名	周彭
学号	181860150
邮箱	1026579097@qq.com
实验时间	40小时+ （编写代码）+ 10小时+ （调试，优化，绘制素材）+ 4小时+ （编写报告）=50小时+

> ⚠提示⚠
>
> 由于本次课程设计代码量较大，一些设计上的细节难以在有限的篇幅内讲清楚，故而在报告中只对大体的框架做一些介绍，项目代码中有详细的注释可以供助教参考。
>
> 如果需要编译代码，请使用 `Qt 5.12` 及以上版本。

第一部分 实验成果

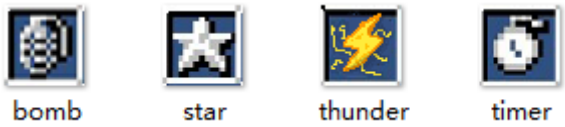
基于第二次课程设计中的 `Battle City` ,使用 `Qt` 中的 `GraphicsView` 框架进行重构,最后实现的功能与效果如下:（注，为了还原原作，游戏中的主要输入方式为键盘，主菜单使用方向键控制“小手”，按下 `A` 键进行选择，如果对操作不熟悉，建议阅读 第四部分 ）

1. 🎮 GamePlay

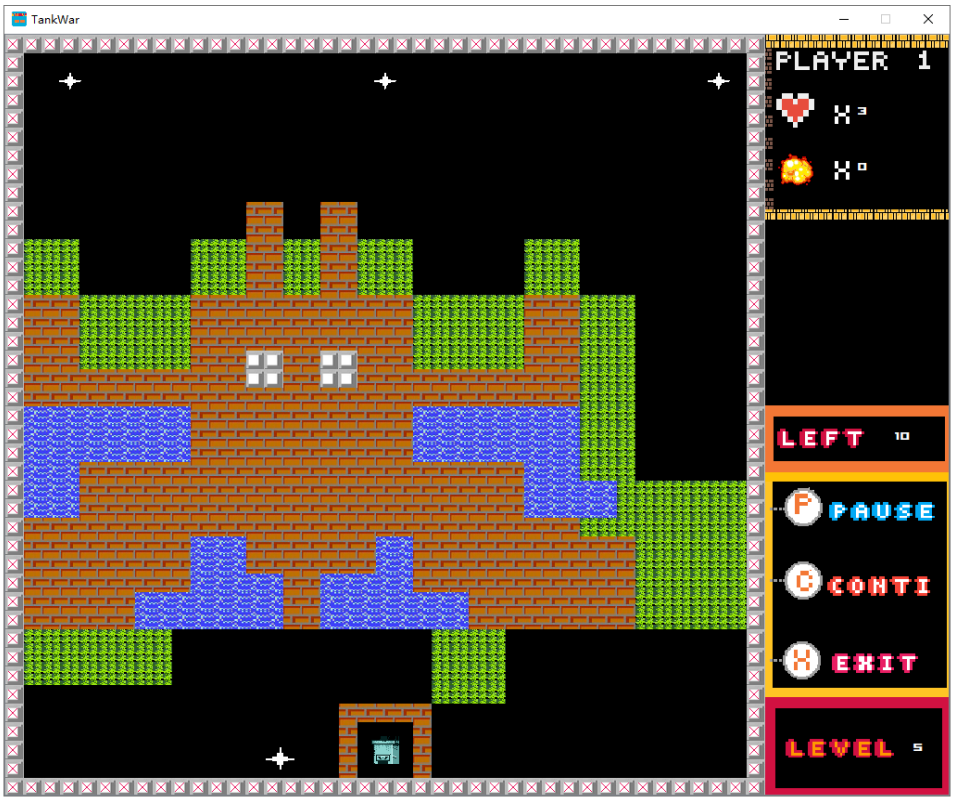


- 内置了7个关卡(自己编辑以及借鉴`Battle City`中的一些关卡设计而成)
- 支持原作中的森林/海洋/砖块/铁墙地形

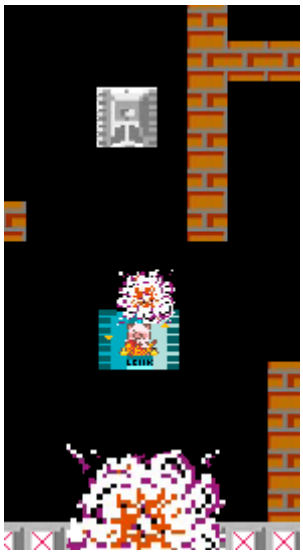
- 还原的道具系统:



- **还原原作中的各种 动画 效果:**
  - 出生:



  - 子弹碰撞动画:



  - 出生无敌效果:

- **丰富的音效：**

  游戏中除了基本的背景音乐（主菜单音乐，游戏中的 `8 bit` 风格音乐）之外，如子弹发射、物体破坏、坦克死亡、玩家胜利以及玩家失败都有对应的音效；同时，**每一种道具**都有独特的音效~~，大大提高了游戏的时髦度~~

- **更强大的游戏AI**

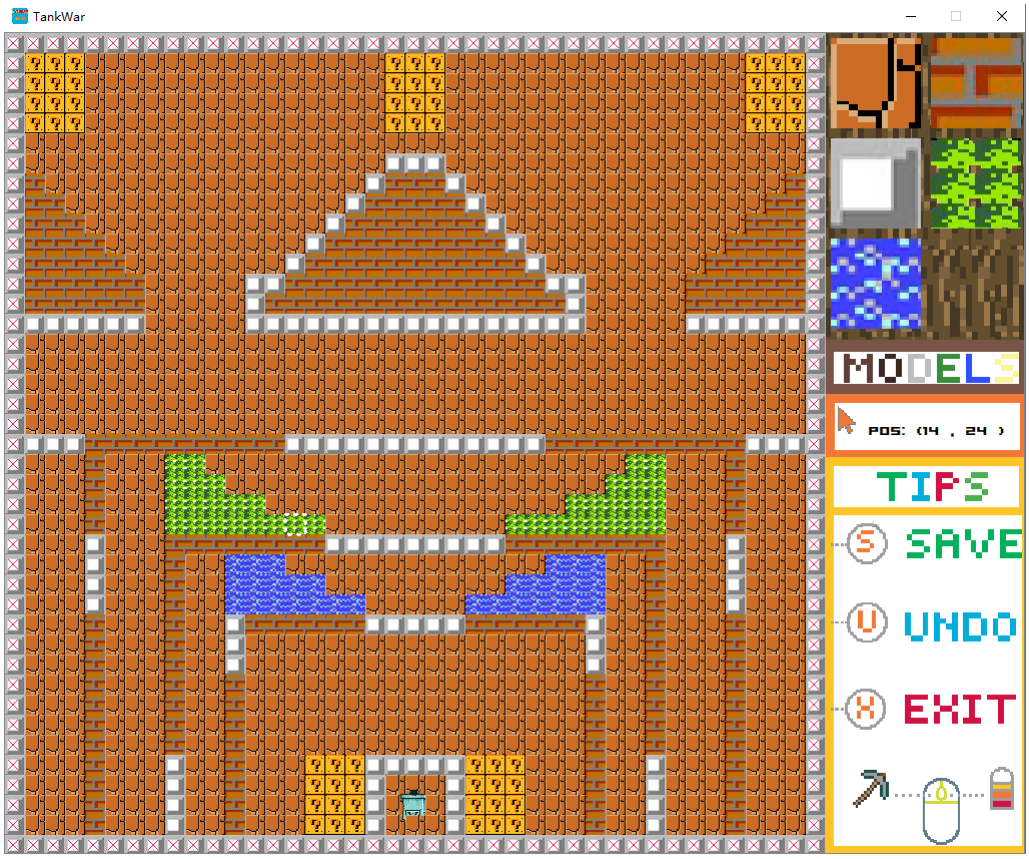
  游戏中的AI采用 `A*` 算法以及 `FSM` 实现，可以对大多数情况作出反应：寻路、游走、在卡住时脱身……同时，每一局游戏中随机刷新基于该机制的不同难度的敌人，让 同一关卡 的反复游玩成为了可能。

- **还有……**

  游戏中可以随时~~时停~~暂停和恢复，妈妈再也不用担心我打游戏不可以暂停了！

2. **坦克制造！**

在这个版本中，我实现了在第二次课设中就想设计的地图编辑器功能，通过主菜单中的 `CONSTRUCT` 可以进入编辑家界面，在该界面中选择 `MAKE` 便可以进入编辑器模式。



该地图编辑器界面如上所示，它支持：

- **点击、拖动——方便你的设计**

  如图上的 `UI` 所示，你可以使用鼠标左键点击右上角的 `模板地图元素` ，在左边的地图区域中建造。你可以对一个地图块进行编辑，也可以按下并拖动鼠标，像画笔一样编辑！除此之外，按下鼠标右键，可以用同样的方式擦除地图上的元素。此外，对鼠标移动到的位置，会有 `光标` 提示

- **出错了？撤销就好**

  无论何时，如果你发现对自己的设计不满意——在不该画的地方画了，或是误擦了，按下 `U` 键，你可以撤销至多6步之前的操作！

- **保存你的设计，游玩你的设计**

  编辑的地图并不是摆设，你可以在 `CONSTRUCT` 模式中游玩自己的设计。当然，和普通模式一样，你可以和你的小伙伴一起游玩！

- **还有……**

  当然，在编辑地图时你也会听到一些有意思的音效，去试试吧！

第二部分 模块设计

2.1 宏观设计

游戏的基于 `Qt` 中的一个绘图框架 `GraphicsView` ，其特点在于**面向对象**的绘图方式——我们需要绘制的每一个图元都是独立的（ `QGraphicsItem` ）。此外，在框架中通过 `QGraphicsView` 类进行显示，`QGraphicsScene` 为我们的舞台， `QGraphicsItem` 需要放入 `QGraphicsScene` 中去显示。这实际上就是一个 `Model-View` 架构。

2.1.1 QGraphicsItem--mapItem--BaseTank,Bullet

由于 GraphicsView 框架的特性，一个自然的想法是， 将坦克、子弹、地形都作为图元去处理，让它们并行地处理各自的逻辑 ，而这些图元本质上都是需要显示在地图上的，我们可以将他们抽象为一个 mapItem 基类。

游戏中的基本地形直接使用 mapItem 进行显示即可，而坦克、子弹则是需要附加一些额外的处理逻辑，故而从 mapItem 基类抽象出 BaseTank 类作为坦克的基类， Bullet 作为子弹类。

此外，为了实现地图编辑器，我基于 mapItem 还抽象出了 EditmapItem 以及 ModelmapItem 表示可以编辑以及模板地图图元。

2.1.2 QGraphicsScene-GameController,MapEditor

只有 QGraphicsItem 还不足够，我们需要一个 QGraphicsScene 来作为它们的舞台，同时对这些 mapItem 进行控制，如 游戏是否结束、生成一些新的图元、删除一些图元等 ，故而需要一个 GameController 类，它继承自 QGraphicsScene 。

除了上述提到的功能， GameController 还需要如下的一些功能：

- 加载地图
- 处理玩家的输入
- 控制绘图

类似地， MapEditor 用于地图编辑器中的 mapItem 的控制与显示。

2.1.3 QGraphisView--GameWindow

最后，考虑游戏中的显示界面，大体上来说，游戏中主要有如下的界面：

- 游戏界面
- 选择界面
- 地图编辑器界面

每一个界面对应于一个 QGraphicsScene ，而它们都通过 QGraphicsView 类进行显示，将这个 QGraphicsView 类抽象成一个类，就是 GameWindow 。

GameWindow 的主要功能在于，进行不同界面的切换显示（更具体的，这些逻辑是通过 Qt 的信号槽完成的），亦即充当一个主窗口的作用。

2.1.4 一些额外的类

由于使用了纯 GraphcisView 框架，没有使用 QWidget 的那一套，故而诸如按钮等控件需要自己实现。体现在本项目中，就是 selectButton 类，其可以对玩家的上下键做出相应，进行选项选择。

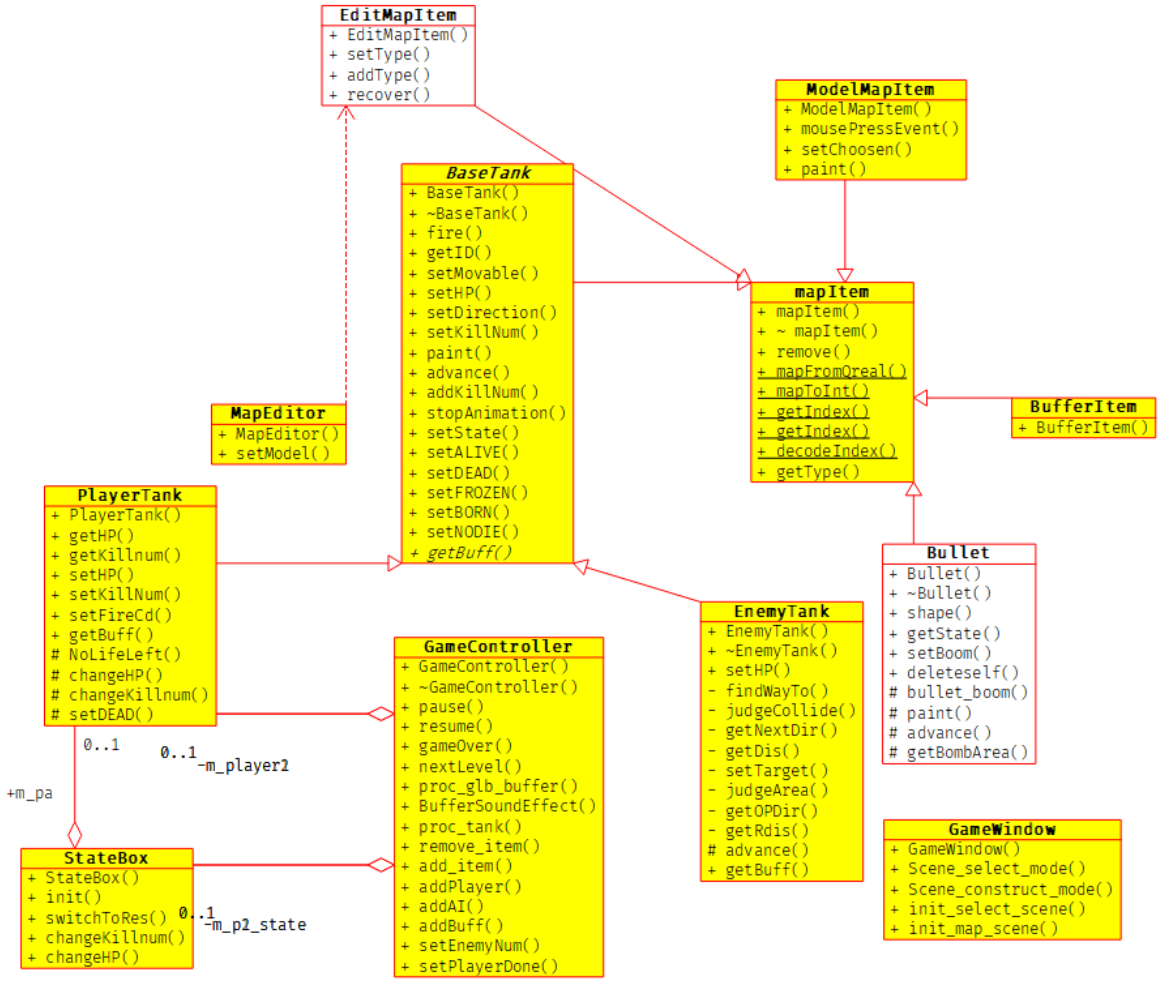
2.1.5 涉及到的宏

游戏中变量中涉及到的宏定义较多，主要可以分为：

- 游戏参。如屏幕大小、一个地图元素大小、游戏的帧率等等
- 一些资源的路径。为了编写代码的方便，将一些资源的路径声明成 const QString 数组，通过特定的编号进行索引，减少了重复工作；
- 游戏中的一些全局枚举类型。如坦克的状态，子弹的状态等等。

2.1.6 类图

为了说明整体类的设计,下面使用 UML 图来具体说明:(如果对图中的箭头不清楚,**可以参见该网站**)



2.2 具体设计——游戏部分

首先大致讲一讲游戏模式的实现，由于无论是普通模式还是建造模式，最终游戏过程的实现是一致的，故而此处以普通模式为例，由于涉及到的方面过多，故而报告以大体框架为主，除去一些必要的部分，不过多深入探讨实现的细节。 具体的接口用途可以参见代码，代码中给出了详尽的注释。

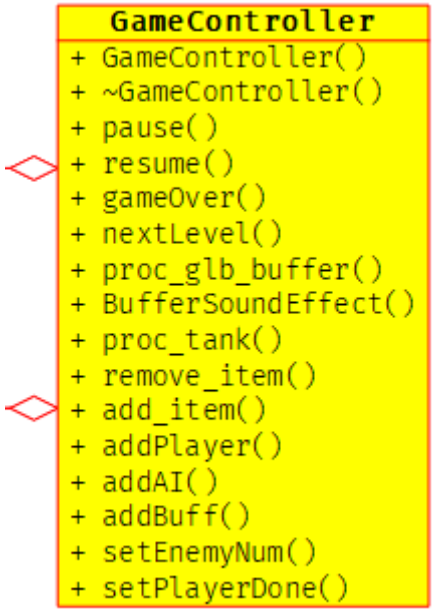
2.2.0 游戏设计

不同于第二次课程设计中使用的**帧循环**实现，在 `GraphicsView` 框架中，并不需要自己显式地去实现一个游戏循环；游戏循环通过 `QGraphcisItem` 自身的 `advance` 函数实现，该函数响应 `QGraphicsScene` 的刷新，从而实现状态的更新。所以，只需要设置 `QGraphicsScene` 的刷新频率，同时使用信号槽将刷新用的 `QTimer` 与 `update(),advance()` 接口进行关联即可。

在这样的架构下，我们只需要完善坦克、子弹的 `advance()` 函数即可。除此之外，游戏中涉及到的各种碰撞的检测，通过 `GraphicsView` 框架提供的 `colliding_items` 可以快速地得到与当前物体碰撞的 `item` 列表，代码可以大大简化。

下面具体介绍各个类的设计。

2.2.1 `GameController` 类



可以看出，`GameController`（以下简称为控制类）的主要作用为：

1. 上层逻辑控制：暂停、恢复、游戏结束、下一关卡
2. 控制游戏全局逻辑：全局道具（如炸弹、时间静止）
3. 多媒体控制：播放道具音效
4. 添加坦克、子弹
5. 初始化游戏

其中，绝大多数函数的实现都十分简单，与坦克类的交互可以通过信号槽机制进行实现，例如当一个坦克死亡时，其发出`tank_dead(int)`信号，通过其参数判断是哪一种坦克，从而执行坦克的复活流程。

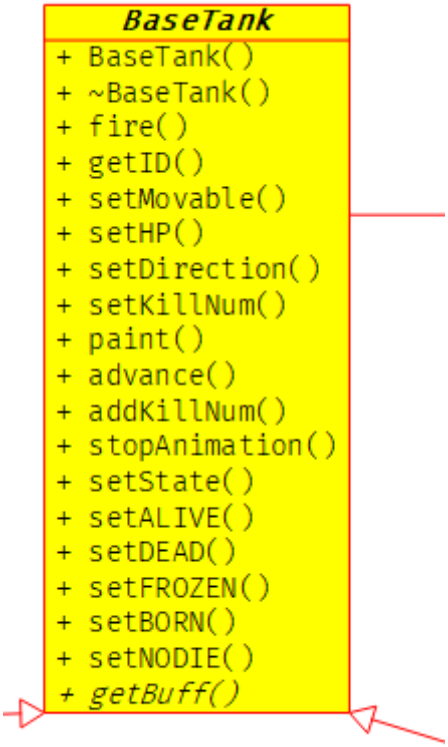
其中，控制类的主要作用在于设置道具以及处理道具的相关逻辑，以及游戏的上层逻辑。

- 道具相关逻辑：

  游戏中一共有四种道具，我设置为10秒刷新一次，为了实现每一轮可以让四个道具均匀出现一次，使用一个`m_buffer_queue`队列来控制当前的道具队列，并将四种道具随机排序后放入队列中，在每一次需要产生道具时获取队首道具；如果道具为空，那么重新加载道具。

- 上层逻辑：
  - 游戏胜利的判定`nextlevel()`：游戏中初始化当前关卡的敌人数量，每当收到敌人坦克的`tank_dead`信号，将该值减少；当该值为0时判定玩家获胜；
  - 游戏失败的判定`gameover()`：游戏中有两种失败条件：如果当前玩家所有生命值归零或者基地被摧毁，那么游戏结束。前者通过`setPlayerDone()`接口实现，后者通过基地被摧毁时发出的信号判定。

2.1.2 `BaseTank`

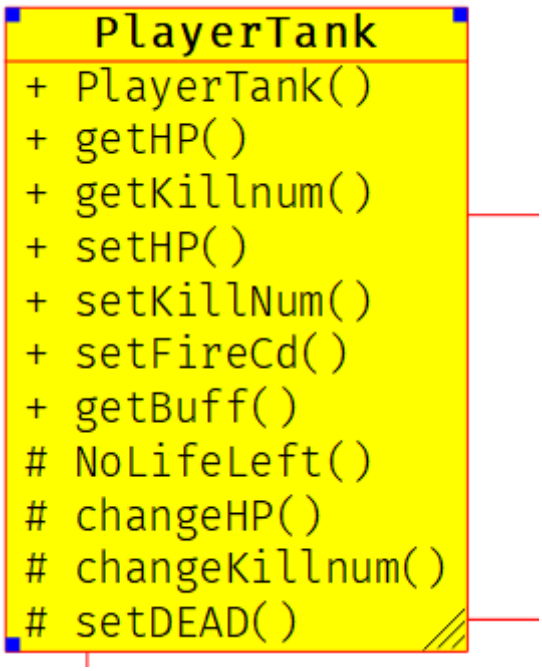


坦克基类，除了重载了 `QGraphicsItem` 必要的一些接口之外，其内部实现了一个简单的状态机，用于判断当前的坦克处于何种状态。该状态机通过 `setState(TankState)` 接口进行状态的变化，也可以通过 `setALIVE()` 等接口进行修改。坦克的状态如下：

- 出生：复活或出生时的状态，1秒后变为无敌状态；
- 无敌：该状态持续2.5秒，不受任何伤害，之后进入存活状态；
- 存活：坦克的一般状态，可以受伤害、死亡、冻结；
- 死亡：当体力降至0以下时进入该状态；
- 冻结：停止移动5秒，之后进入存活状态

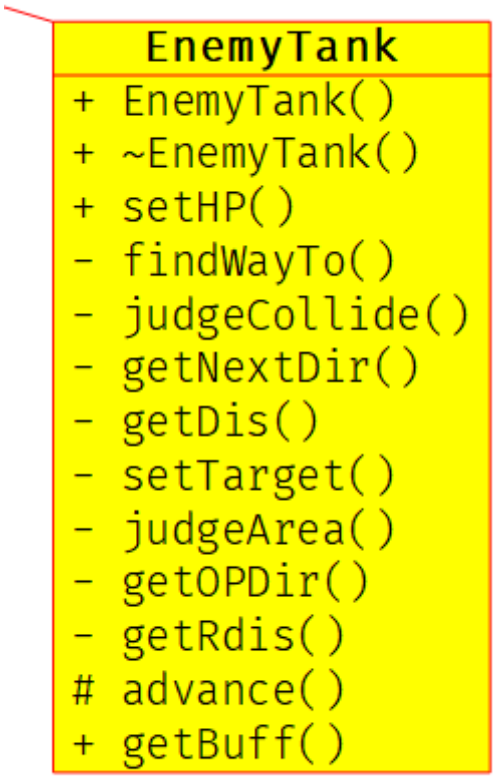
除了该状态机以外，代码中还实现了一系列的 `get/set` 方法，包括 `id` 获取（判断是敌方还是玩家坦克）、设置当前方向的接口。

基于该基类，实现了 `PlayerTank` 类：



相较于基类，主要在于添加了`getABuff`函数用于处理道具的相关逻辑。

敌方坦克实现相对而言较为复杂，内部使用`A*`算法实现寻路，对外接口与`PlayerTank`没有太大区别：



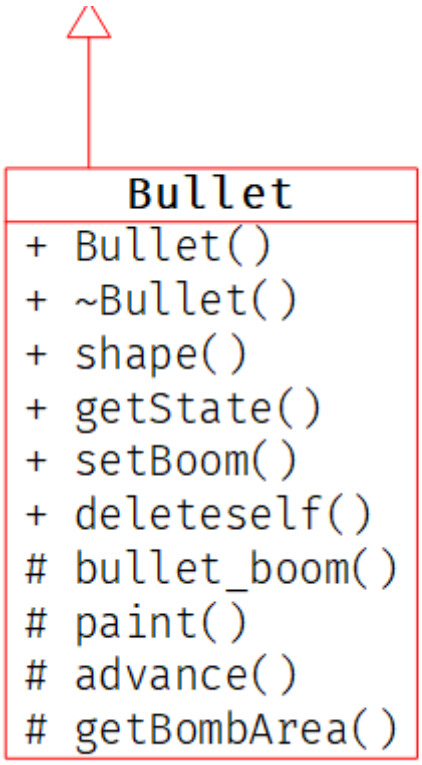
其内部接口参见注释：

```
1       /* 用于寻路的相关接口 */
2       void findWayTo(Item_type type=BASE);// 寻路主函数，目标为type，例如基地，道具
3       bool judgeCollide(Dir temdir);      // 判断当前位置是否会撞到物体
4       Dir getNextDir();                   // 根据已经规划好的路径进行方向调整
5       inline int getDis(int sx,int sy,int dx, int dy);// 用于寻路算法，获取两个点之间的
    哈密顿距离
6       void setTarget();                  // 设置当前的局部目标点
7       inline bool judgeArea(const int& i,const int& j,const Dir& tdir,const
    Item_type& type)const;// 判断区域是否可达
8       Dir getOPDir(Dir);                 // 获取反方向
9      inline  qreal getRdis(qreal,qreal,qreal,qreal);// 获得两个点之间的实数距离
```

`A*`算法的实现在此处不做展开，大体上可以概括为将地图网格化、计算每一个地图节点的权值等操作，具体算法的实现并不是本门课程的核心内容，故而在此不多做展开，如果助教感兴趣可以参见代码。

2.1.3 `Bullet`类

子弹类继承自`mapItem`：



`shape(),paint()`等函数用于重载`QGraphicsItem`的相关接口，其中比较重要的接口为内部接口`move()`，其提供了`Bullet`碰撞检测的逻辑：

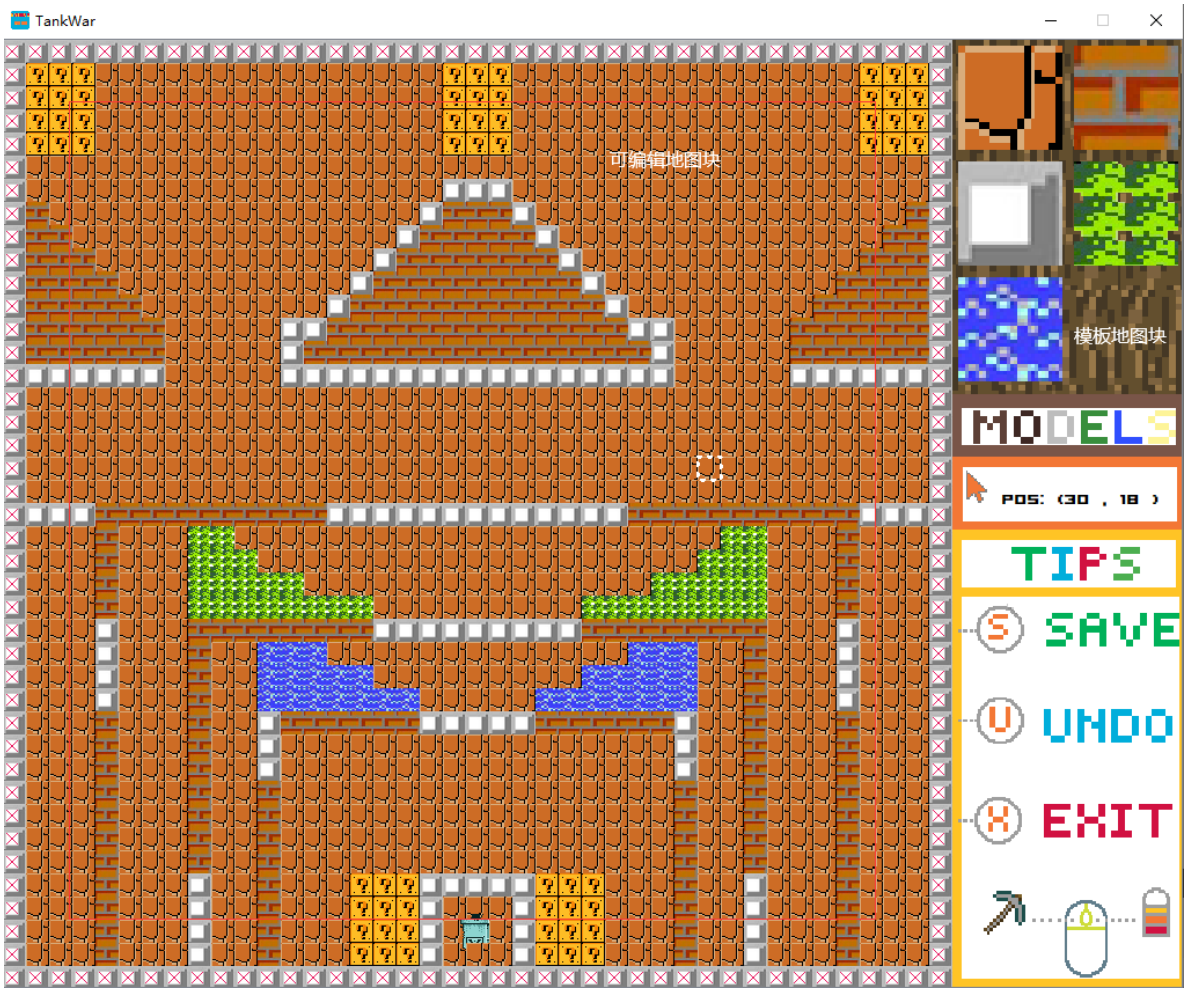
- 如果碰到了当前子弹伤害可以摧毁的物体，则将其摧毁；
- 如果碰到了子弹，将子弹设置为爆炸（通过`setBomb()`接口）

值得注意的是，子弹的爆炸范围与其自身的体积不同，故而使用`getBombArea`接口用于根据当前方向获取爆炸范围，设置为1*3的爆炸范围。

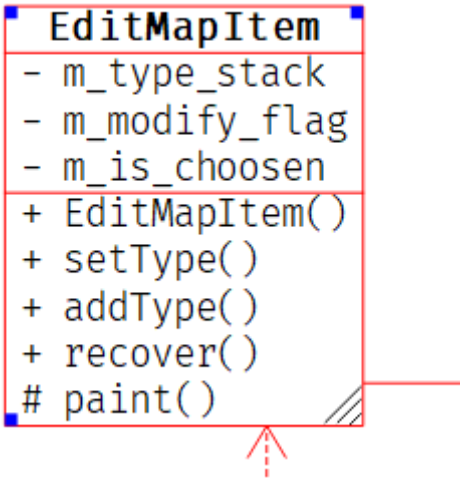
此外，道具类`BufferItem`、`selectbutton`等类没有做详细介绍，但是代码中都有详细的注释，其实现也是较为易懂的，在此不多做阐述。

2.3 具体设计——地图编辑器

地图编辑器通过`MapEditor`类进行控制（以下简称为控制器类），下图中左侧的地图元素为`EditmapItem`，右侧的地图元素为`ModelmapItem`。



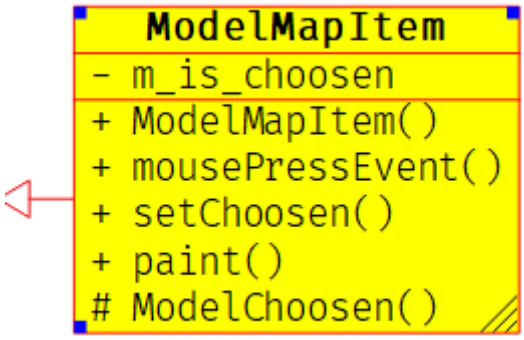
2.3.1 EditmapItem



可编辑地图块继承自 `mapItem`，其为上层提供了修改当前地图块类型的接口；为了实现撤销操作，其内部实现了一个记录状态的栈，并通过 `recover()` 接口恢复到栈顶的状态，通过 `addType()` 接口向状态栈中加入一个新的状态。

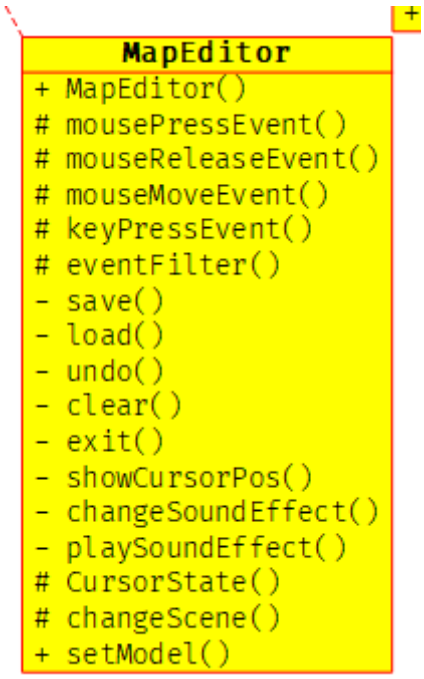
由于地图中有一些地形是不可以编辑的（如基地，坦克的出生地），故而 `m_modify_flag` 用于标记当前的地图块是否可以编辑。

2.3.2 ModelMapItem



`ModelMapItem` 同样继承自 `mapItem`，其通过鼠标按下事件来使得自身被选中，从而在地图编辑器中使用该地形作图，`m_is_choosen` 用于标记当前是否被选中。

2.3.3 `MapEditor`



地图编辑器继承自 `QGraphcisScene`，其内部记录了地图编辑器中可以编辑的 `EditMapItem` 的 `QList`，以及模板地图块，还有用于撤销操作的一个队列。

此外，地图编辑器重载了鼠标按下事件，用于完成一个单击事件（左键用于绘图，右键用于清除）；重载了鼠标拖动事件，用于完成拖动绘图。在上述过程中，我们记一次绘图为一个基本单位，以该单位完成撤销操作，故而需要重载鼠标按键松开事件，记录一个绘图事件的完成。

至于其他的一些操作，如保存、退出，实现都较为简单，详尽的内容可以参照代码。

第三部分 与前两次课程设计的关系

第一次课程设计我基于 `easyx` 图形库完成了一个简单的贪吃蛇，第二次则是完成了坦克大战的控制台版本，一路走来，我都是以做游戏为主。当然，同样是做游戏，我对于游戏整体的架构也有了更深的认识。

一开始的贪吃蛇制作，我只是用了一个控制类以及两个角色类，导致了游戏整体的可拓展度不高；到了坦克大战，我将 `OOP` 思想进一步延伸，将游戏中的诸如地图、坦克等类进一步细分，并阅读了关于游戏设计模式的一些书籍，对“帧循环”下的游戏设计模式有了进一步的了解。

此次游戏设计，借助`GraphicsView`这一强大的框架，我得以施展拳脚，在图形显示方面不再受拘束的情况下，专注于`gameplay`的设计，同时也初步涉及多线程下的编程中的一些问题，以及如何使用面向对象的思想进行图形化编程。

第四部分 操作手册

- **菜单界面**
  - 按方向键上下移动光标选择模式
  - 按下`A`选择当前模式
- **游戏中**
  - 玩家一: 使用方向键控制, `/`键发射;
  - 玩家二: 使用wasd键控制,空格键发射.
  - 作弊：游戏中，按下`O`即可无限火力；按下`K`可以跳关；按下`N`可以短时间无敌；按下`Z`将当前命的体力加满
- **地图编辑器**
  - 鼠标左键：绘制、选择当前的模板地图块
  - 鼠标右键：擦除
  - 保存：S键
  - 撤销：U键
  - 清除：C键
  - 退出：X键

第五部分 困难与应对

此次课程设计是三次课程设计中耗时最久的，之前的两次设计我基本上都可以提前一至两周完成，而此次课程设计我足足花了三周多，一方面是第一次使用`GraphicsView`框架，上手用了一定的时间；另一方面，游戏的运行实际上是多线程的，遇到一系列奇奇怪怪的bug。

1. **段错误，还是段错误**

   由于游戏中需要生成大量的动态对象，故而何时删除动态对象成了一个十分棘手的问题，如果不加注意，要么会产生内存泄漏的问题，要么就是段错误。尤其是在`GraphisView`框架下，实际上每一个`QGraphicsItem`都是并行运行的，故而往往你`delete`它之后，它可能还有方法在执行，这时候就会抛出段错误。最后将所有使用`delete`的`QObject`派生类都使用`deletelater`进行替换，从而解决了该问题。

2. **qt creator**

   这是一个神奇的`ide`，它的`debug`模式相比`vs`有许多不方便的地方，不过在上手了之后发现了其诸如`TODO`提醒、自定义注释等功能还是挺实用的。

3. 素材问题

   网上可以直接使用的素材其实不多，最终的基本游戏素材（如坦克等）是我从网上找到的资源，而诸如ui等（除了开始界面的背景），基本上都是我自己用`pixilart`网站画的（所以看起来有点挫）。

总而言之,言而总之,此次的课程设计,痛并充实.