QT 大作业《TankTrouble》项目报告

小组：不足为道，且自斟酌

王梓沣 2200013092，伍羿寰 2200013181，沈千帆 2200013220

0 目录

1 概述

1.1 代码结构

本项目使用多文件编写，包括

图 1.1.1 界面

头文件	功能
AllWindow.h	对所有界面进行统一调用
HintWindow.h	显示指引界面
EnterWindow.h	进入界面，游戏的主菜单
MenuWindow.h	游戏的暂停菜单
Music.h	音乐播放及调整声音的界面
TankTrouble.h	项目核心，游戏界面
EndWindow.h	结算界面

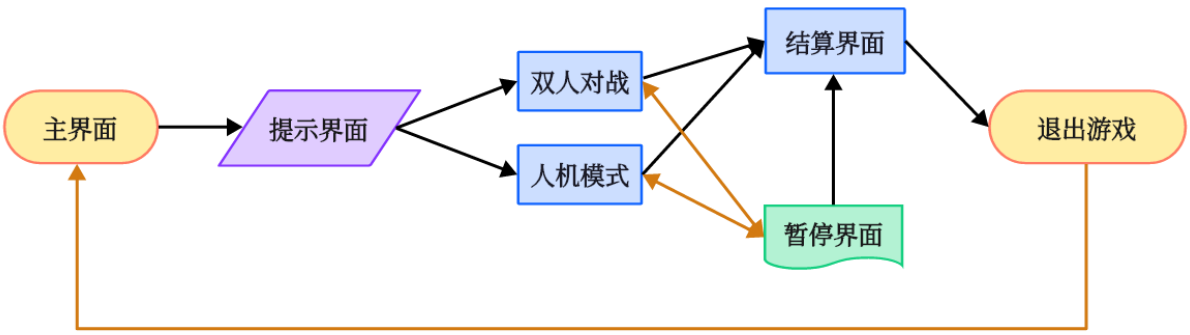
图 1.1.2 内核

头文件	功能
Object.h	物品的基类
Wall.h	墙体实体的子类
Bullet.h	各种子弹类型的子类
Tank.h	可操控坦克的子类
ToolBox.h	道具箱的子类
Modify.h	处理碰撞的工具类
painting.h	绘制游戏进程的工具类
CreateWall.h	创建墙体的工具类
paintingExplode.h	绘制爆炸效果的工具类
config.h	宏定义，表征各实体的大小等

各个代码部分采用 qmake 进行组织和编译，运行在 qt6.4.0 上。

1.2 游戏结构

本游戏提供了“人机对战”与“双人对战”两种模式供游玩。进入任一模式后，将有按键教学帮助玩家了解坦克操纵方式。游戏的整体结构如图所示。



1.3 游戏主要功能

1.3.1 初始化

一局游戏开始后，程序将随机生成一幅保证全连通的新地图。同时，随机生成双方坦克的位置与角度，保证游戏的随机性与可玩性。

1.3.2 运动与开火

玩家可通过 ESDF 或上下左右操纵坦克的前后移动，顺逆时针旋转，通过 Q 或者 M 操纵坦克开火。

坦克共有五种行动方式：前进、后退、顺时针旋转、逆时针旋转与开火。开火后，子弹将飞行一段时间，期间撞上墙壁将反弹，撞上任意一方坦克将击毁之。如没有击中任何目标，子弹一段时间后将消失。

如一方坦克被击毁，游戏还将进行 5 秒。期间如另一方也被击毁，则该局视作平局。

1.3.3 道具

我们设置了三种道具：霰弹枪、冲锋枪与破片地雷。

- 霰弹枪：共一发弹药，一次性向前方一定扩散角度发射十发子弹。
- 冲锋枪：共二十发弹药，以更快的频率发射较小的子弹。
- 破片地雷：共三发，开火后在原地布置一枚地雷，布置完成三秒后，地雷开始工作。若地雷被坦克或子弹触碰，则会爆炸，向四周发射破片。

游戏进程中，每过一段时间，会在地图上随机位置生成一个道具箱。坦克触碰道具箱后将获得对应能力，即开火方式由普通子弹变为对应道具，道具使用后恢复。

1.3.4 暂停和音量控制

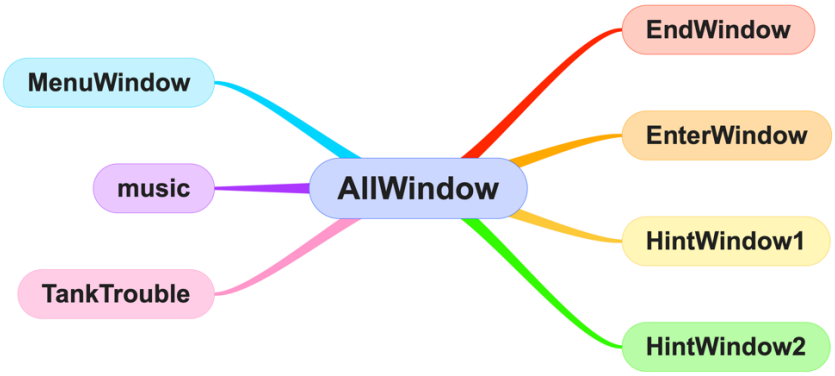
可在游戏任何界面进行背景音乐音量调节。游戏进程中，可点击右上角暂停键。之后可选择返回游戏或回到主界面。

1.3.5 结算

任意一方得分达到 4 后，游戏结束，进入结算，展示双方胜负关系与具体比分。

2 项目模块介绍与类设计细节

2.1 界面



AllWIndow 类中需要用到其余七个分支类，实现所有界面之间的串连。

2.1.1 EnterWindow

是游戏的开始界面，包含开始界面的按钮变量和绘制函数。

1. button_volume：调节声音的按钮。
2. button_start1：选择人机模式的按钮。
3. button_start2：选择双人对战模式的按钮。
4. button_exit：选择是否退出游戏的按钮。

包含以下函数：

1. paintEvent：绘制背景图。

2.1.2 HintWindow

分为 HintWindow1 和 HintWindow2。分别是人机和双人对战的提示界面，用于进行操作的指引。

人机模式的提示界面包含以下变量：

1. button_volume：调节音量的按钮。
2. QPushButton* button_return：返回主菜单的按钮。

包含以下函数：

1. paintEvent：绘制背景图。
2. keyPressEvent：设置键盘和提示画面出现的关联。
3. drawKeyPressedImage：使得按到相应的字母按钮变红以达到提示效果。

双人对战模式的提示界面包含以下变量：

1. button_volume：调节音量的按钮。
2. QPushButton* button_return：返回主菜单的按钮。

包含以下函数：

1. paintEvent：绘制背景图。
2. keyPressEvent：设置键盘和提示画面出现的关联。
3. drawKeyPressedImage：使得按到相应的字母按钮变红以达到提示效果。

2.1.3 MenuWindow

是游戏中的菜单界面，在点击暂停按钮时显示。包含以下变量：

1. return_game：继续游戏的按钮。
2. game_over：直接结束游戏的按钮。

2.1.4 EndWindow

游戏的结算界面，会显示当前比分和胜方。

包含以下变量：

1. red_score：红方目前的得分。
2. blue_score：蓝方目前的得分。
3. mode：游戏模式。
4. exit：菜单界面的按钮。

2.1.5 Music

负责音乐播放的类，用于背景音乐效果的播放。

包含以下变量：

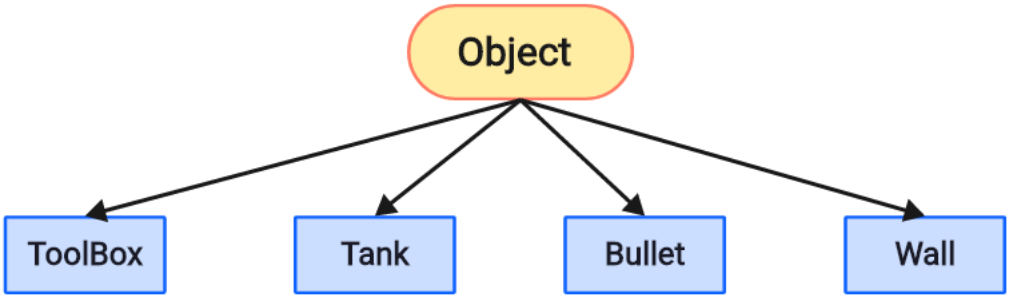
1. musicplayer1：控制音乐的播放。
2. volumeslider：调节音量的滑块。

3. label_red：图标。
4. label_blue：图标。
5. button_silence：静音/打开声音的按钮。
6. clickcount：点击该按钮的次数，已决定是静音/打开声音。

2.2 内核

2.2.1 游戏的物品类 Object 与其子类

Object 类与子类的关系如图所示：



Object 类是所有游戏内实体物品的基类，用于对碰撞，反弹等各类物理效果进行处理。包含了所有物品公有的成员变量，如下所示：

```
static vector<Object*> object;   //所有物品的集合
int seta;                        //多边形旋转的角度
int x, y;                        //矩形左上角的坐标
string name;                     //指向的派生类名称
bool alive;                      //物品是否还存在，每帧结尾清扫
QPixmap picture;                 //Qt图片资源文件
QPolygon polygon;                    //碰撞箱
```

Object 类的函数如下，用于处理 object 共有的行为。

```
Object(string name,int _x=0,int _y=0,int _seta=0);
QPolygon Transform();                       //处理旋转多边形的姿态变化
bool if_crash(Object* target);              //碰撞检测
virtual Modify collision(Modify _modify,Object* target)=0;
//纯虚函数，对不同组合的碰撞做出不同反应
virtual void explode();                 //多态，碰撞后物品的删除
virtual void forward();                 //多态，完成Tank和Bullet的移动
virtual void backward();
double deg_to_rad(int x);                   //后端计算，用于角度和弧度的相互转换
int rad_to_deg(double x);
int Manhattan(Object* target);  //计算多种物品间的距离，用于减少判断次数，优化性能
virtual ~Object();
```

下面说明各个派生类的特有函数和变量。

(1)Tank：public Object

Tank 类的成员如下

```
int bullet_count;                   //当前还剩余的弹药数目
```

```
int tank_color;                    //Tank的阵营
int state;                         //当前Tank所持有的道具类型
int ai_rotate, ai_move, ai_fire;//人机算法的行为信息
int target_x, target_y;
QMediaPlayer* vfire;          //导入开火的声音资源
QMediaPlayer* vmgun;
```

Tank 的自有成员函数如下

```
void fire();                  //Tank开火的状态，包含以下三个函数的调用
void fire_mine();
void fire_shotgun();
void fire_machine_gun();
void clock_rotate();          //Tank的移动行为
void anticlock_rotate();
void AI();                    //AI算法的实现
void Random();                //AI的随机重启
int Rand();                   //随机数
bool pd_move(int dx,int dy,int dseta);   //判断 AI 的移动合法性
```

对于 AI 的解释：

我们实现了人机坦克的移动、旋转、移动目标选取、同格攻击与随机重启。下作详细介绍。

人机坦克倾向于从格子的正中心移动到另一个格子的正中心。每次达到一个目的地后，人机坦克会检测周围四个方向的可达目标，设置其为下一个目的地。人机坦克会优先选取没有走过的格子作为目的地。向目标移动分为“转向”与“前进”两个部分，坦克会先判定自己坐标与目的地坐标的倾角是否与当前姿态倾角一致，如不一致则优先旋转调整，旋转会选择正逆时针中最快的方式。倾角一致后，坦克将调用 forward 前往。

移动过程中，一旦人机坦克判定到自己和玩家坦克处在同一格子内，将立即调整倾角并开火。

若移动过程中出现错误，如卡墙等，人机坦克将调用 Random 接口，选择一种不会卡死的操作方式（包括前进后退，正逆时针旋转），脱离当前状态，再选取下一个目标。

(2)Bullet：public Object

Bullet 处理的是子弹的移动，反弹、销毁和自动消失。拥有以下成员变量

```
int lifeTime;        //子弹的存在时间
Tank* p;             //子弹的来源，用于记录上限
int kind;            //各种子弹的类型记录
const QString des[3]//子弹资源文件的地址
```

子弹的反弹，销毁等行为都通过 Object 的成员函数多态实现

(3)Wall：public Object

用于储存建设的墙体，没有新增函数和变量。其使用详见 **CreateWall.h**

(4)ToolBox：public Object

用于储存生成的道具箱，并且有随机生成道具箱的函数。

```
void generate(int& toolCounter, int& num);
```

该函数在 update 中调用，每隔一定时间在场景中随机生成各类道具箱，上限是 6 个。

2.2.2 工具类 CreateWall，Modify，painting 和 paintingExplode

工具类用于实现游戏主场景所需的函数。简述如下：

(1)CreateWall

CreateWall 是构建墙壁的算法类。CreateWall.h 中所需变量如下：

```
struct Build_Wall
struct Block
```

这是创建随机地图时所用到的封装。使用一堵墙两边的格子来表征该墙。

```
pair<int, int> Coor_Block(int i, int j);
void test_wall();//
void Create_Wall();
extern bool vis[Horizon_Block_Number][Vertical_Block_Number];//格子是否加入生成树
extern bool dead_wall[Max_Block_Number][Max_Block_Number];//墙是否被拆除？
extern vector<pair<Block, Block> >wall_que;//当前可以拓展的边
```

使用生成树算法进行墙的创建：首先在所有Block之间生成墙体，然后进行墙体的拆除，直到地图实现全联通，则创建结束。

(2)Modify

修改操作的封装。同一时刻，对同一物品的操作可能有很多（包括前后移动，左右转向，发射子弹与销毁等），使用 Modify 统一存储，最后一起实际修改到物品上。包含以下变量：

```
int x, y;
int seta;
int state;          //本次修改给物品带来的姿态影响。
bool if_modify_seta;
bool if_explode;
bool if_move;
int if_modify_state;//本次操作后，各种事件是否生效的判定，便于回退或进行下一步操作。
```

(3)Painting.h

是游戏过程中对资源进行绘图的函数，只由 painting.h 这一文件组成，其中函数为 inline。文件中只有一个函数，即

```
void paintingImage(Object* obj, QPainter& painter, int& flagColor)
```

这个函数会对当前存在的所有 Object 使用，绘画出地图上所有物品的当前状态。

(4)PaintingExplode

这是实现爆炸动画的工具类，对于当前帧应当销毁的物品，处理如下：

先记录需要销毁的物品位置，然后在这一位置生成一个 paintExplode 对象，在 update 中播放这一对象爆炸的动画，然后销毁这个对象。其成员变量是播放这些动画所需要的。

```
int m_x;
int m_y;
bool if_explode;
```

```
    int explode_record;
    int m_index;
    std::vector<QPixmap> source;
```

这个对象有唯一的成员函数，即 `void updateExplode();`用于把需要播放动画的对象同步给主界面。

2.2.3 游戏界面类 TankTrouble

游戏界面类是整个游戏逻辑的核心，它将内核部分和界面部分的逻辑整合，并对玩家的键盘和鼠标输入进行处理。下面对这个类进行详细解释：

（1） 成员变量

```
    vector<paintExplode*> toExplode{};      //将要爆炸的物品记录
    bool isAliveA = 1;                      //A，B坦克的存活情况
    bool isAliveB = 1;
    int Gamemode;                           //游戏模式 (0:PVC 1:PVP)
    QPushButton* button_volume;          //游戏中调节音量和暂停按钮
    QPushButton* button_menu;
    int counter = 0;                     //处理游戏结束的计时
    int bluescore = 0;                   //双方分数
    int redscore = 0;
    int fireCounter = 6;
    int toolCounter = 0;
    int toolNum = 0;
    int fireInterval[10]{ 7,7,1,2,2 };      //处理开火的时间间隔
    QMediaPlayer* vboom;                 //爆炸的音效

    QTimer* timer = new QTimer(this);       //全局计时器
    bool keyPressFlag;                   //处理按键
    struct {}key0;
    struct {}key1;
```

（2） 成员函数

```
    TankTrouble(QWidget* parent = nullptr);
    ~TankTrouble();
    void keyPressEvent(QKeyEvent* event) override;       //接收键盘输入
    void keyReleaseEvent(QKeyEvent* event) override;
    void GameStart();                                    //游戏的开始
    void DoOperation();                                  //处理键盘输入和AI输入
    void updateAll();                                    //将更改同步到界面
    void draw_score(QPainter& painter);                      //绘图 绘出计分板
    void update_score(int new_bluescore, int new_redscore); //向计分板同步分数

protected:
    bool GameEnd(int blue, int red);                         //游戏结束
    void paintEvent(QPaintEvent* event) override;            //绘制游戏界面
```

每次游戏开始时，先调用GameStart对局面进行初始化，接收到外界输入后，在updateAll函数中进行处理，实现了整个游戏的逻辑循环，并调用paintEvent把后端的修改显示在图形界面上。从而实现了整个游戏的可视化。

3 组内分工情况

我们将项目实现划分为三个部分：后端、交互与界面设计。

后端（伍羿寰）

后端负责实现交互所需要功能的接口，具体如下：

1. 物品（坦克、子弹、墙、道具及道具箱）的放置，碰撞，移动等。
2. 地图的随机生成算法与人机行为算法。

交互（王梓沣）

交互负责整合玩家操作、后端接口与界面素材，并绘制游戏进程反馈给玩家。具体如下：

1. 实现玩家对键盘操作与后端接口的链接。
2. 绘制物品，生成可视画面。
3. 将后端算法与界面设计的相关内容整合为可运行项目。

界面设计（沈千帆）

1. 设计并实现游戏各界面。
2. 搜寻贴图、音乐音效等素材。

4 项目的总结与反思

本次项目可以说是我们合作实现较高码量项目的“处女作”，实现过程中，我们遇到了很多问题并尝试了各种方式去解决，收获了很多单打独斗所无法获得的经验，也产生了很多相应的教训。下作总结：

经验与收获

1. 任务拆分要合理。多人合作，最紧要的难题在于“交流”，从 idea 到具体实现思路，再到代码，不同人的理解总会有语言一时无法表述清楚的区别。如果任务划分不明确导致耦合度过高，就会严重拖慢开发进度，导致项目无法如期完成。本次项目伊始，我们便首先将游戏逻辑和界面美术拆分出来，因为我们认为，这两个 part 可以并行开发而无太多冲突。然后，我们将游戏逻辑拆分为底层算法与交互合并，交互负责统合底层算法、界

面与玩家操作，从而让相关任务尽可能达到“并行处理”，同时也消减了大量不必要的交流。开发过程中，底层与界面互不干扰，只需将代码逻辑向交互说明即可。我们认为，这是本次项目实现如期完成的重要保证之一。

2. 时间规划要明晰。人或多或少有拖延偏向，如若单打独斗倒也罢了，但在团队合作中，一个人的拖延将不可避免地影响项目总的推进。我们提倡少而精的共同交流与细密互通的环节交流，即任务之初的第一次会议，在确定项目选题，大题框架与分工之外，还要定下一个时间表，明确何时要完成何事，下次会议项目要推进到什么程度。时间表可以调整但不能没有，否则很容易学陷入无穷的 ddl 地狱，最后疯狂赶工。时间表确定后，如项目实现存在任务耦合的，保持畅通的信息交流，包括常量设置、接口命名等，以减少后期倒改“远古”代码的惨案。王梓沣向底层提供的 to do list 便极大提高了底层工作的效率，避免了可能发生的很多矛盾。
3. 好的 idea 剩过无效代码的堆积。在算法考试中，一时的灵光乍现可能剩过数小时的无意义敲键盘，在项目效果展现中同样如此。一个好的 idea，也许用不了多少码量，但却能让用户耳目一新，眼前一亮，效果远胜繁复却平凡的普通功能。例如本项目的教学界面，沈千帆巧妙地将按下的对应键变红，极其明确而灵动地向玩家演示了按键操作，效果极佳。
4. 我们学习到了很多新工具，如 QT，git，github 等，这对我们未来的项目开发很有帮助。

教训

1. 代码备份要牢记。我们都是初次接触 git，使用 git，操作上难免会有疏漏，所以及时的打包与本地、u 盘、微信备份仍然是必要的。第一次使用 git 上传时，我们发现项目文件上传失败后全都变成了已编译的 obj 文件，且该操作无法回退。很不巧，我们并没有及时备份工作进度，于是不得不从十二小时前的老版本开始重新制作，干到半夜三点。
2. 代码风格要有规矩。写代码不能想当然，变量名不能随心所欲，中英夹杂。类名与函数名在大小写问题上要统一，否则会引发持续的理解误差。底层逻辑相关代码起初的风格就极其不佳，后虽做了一定调整，但有些已难再调，留下了很多隐患。