# 超级马里奥游戏项目介绍



## 概述

## 游戏进程概述

- 开始游戏界面
- 用户按下"S"键，即可开始游戏；
- 游戏操作说明
    - 用户使用方向键控制mario移动，空格跳跃，“B”键可发出火球攻击；
- 游戏过程
    - 用火球攻击敌人或者正中踩中敌人，即可消灭敌人，否则产生其它碰撞mario会受伤或者死亡；从正下方撞击道具箱即可出现道具；
- 结束游戏

        - mario死亡，结束游戏；
- 通关游戏
        - 当用户成功通关，进入下一关。

## 环境

QT(Assitant 6.24-MinGW 11.2.0 64-bit)

Visual Studio 2022

## 类名及属性概述

### 类-概述

- Terrain 地形土地块
- Block 墙砖
- Fireball 火球
- Mario 马里奥
- Pipe 管道
- Enemy 敌人
    - Goomba 蘑菇敌人（贡巴）
    - Turtle 乌龟敌人
- Box 道具箱
- Flower 花
- RedMushroom 红蘑菇
- GreenMushroom 绿蘑菇

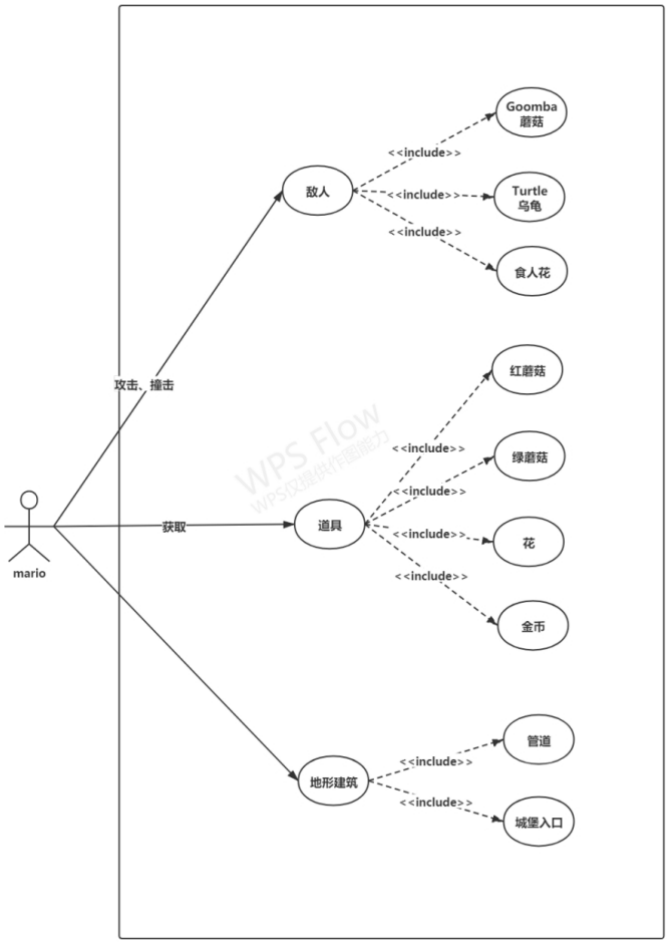
### 属性概述

- walkable 【属性】是否可走
- colidable 【属性】是否可碰撞
- moving_speed 【属性】移动速度

# 设计思路

## 对象设计

### **Mario**用例




- 攻击碰撞敌人
- 获取道具
- 触发地形机关
- 对体系结构重要的用例
    - 攻击敌人
      简短描述：此用例不是马里奥死就是敌人亡。此用例的参与者为：mario、继承enemy类的敌人实例。
    - 获取道具
      简短描述：此用例改变马里奥的状态，（如生命值、大小形态、是否能喷火球）。此用例的参与者为：mario、藏在box里面的道具类实例（Flower【花-----吃了能喷火】、mushoroom【红蘑菇----Mario变大、、、绿蘑菇---Mario生命值加一】、coin【金币】）。
    - 地形建筑
      简短描述：此用例让mario在特定的背景建筑的碰撞下能够产生特定机关效果。此用例的参与者为：Mario、地形。

## 对象交互部分

利用状态机模型，将游戏中的可交互对象抽象为拥有不同属性以及不同状态的状态机。在对象与对象产生交互的过程中，即是通过调用状态转换函数将各对象的状态切换，从而完成交互的过程。

## 游戏画面显示

通过QT内置类QGraphicsPixmapItem(被游戏对象大类Object集成)像素图加载类，其中集成了图层加载，抗锯齿，动态模糊，碰撞箱，碰撞检测等功能)中的animate()(像素图刷新)方法，通过游戏运行过程中动态维护的各对象状态加载图层，如下提供了Mario 类的animate()方法：

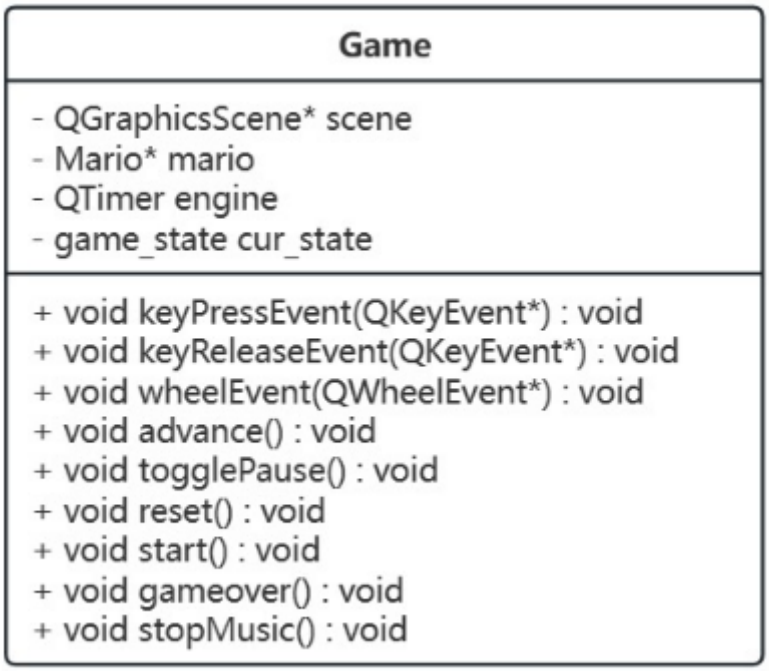
```
void Mario::animate()
{
    // handle bouncing 处理强壮状态
    if(jumping == false && bouncing){...}

    // handle phantom mode 处理幽灵状态
    if(phantom){...}

    // handle transformation 处理转换过程
    //(在吃了强化蘑菇后，会进入由正常状态到强壮状态转化的过程)
    if(transforming){...}

    // set proper texture 刷新转化动画
    if(transforming){...}
    else if(dying || dead)
        setPixmap(texture_dead);
    else if(moving && !jumping && !falling)
        setPixmap(texture_walk[big][(walk_counter++/(running ?
running_div : walk_div))%3]);
    else if(jumping || falling)
        setPixmap(texture_jump[big]);
    else
        setPixmap(texture_stand[big]);

 /// 状态刷新
 ////////////////////////////////////////////
 //检查是否通关
 if(successing){...}

 //检查是否为下落状态
 if(falling){...}

 //检查是否为进入管道状态
 if(piping1 && piping){...}
```

```
    //处理下落过程
 if(piping2 == true){...}
 ////////////////////////////////////////////////

    // mirror the texture along the horizontal axis if needed
 //镜像，坐标对称
    if(dir == LEFT)
        setPixmap(pixmap().transformed(QTransform().scale(-1,1)));
}
```

> 在本例中，*animate()*方法通过检查每一个可能的状态，然后加载画面

## 游戏主窗口概述

### Game类




继承于QGraphicsView相当于创建一个用于显示内容的窗口

数据成员scene:继承于QGraphicsScene相当于在窗口中创建一个用于绘制各对象的画布

方法：

- `keyPressEvent()` 以及 `keyRelease()` 按键侦听：处理如按下/松开键盘左右键上下键以及空格等键盘事件

- `wheelEvevnt()` 鼠标滚轮事件，在鼠标滚轮滚动时会放大/缩小屏幕。
- 以及 `reset()` 重置游戏 `start()` 开始游戏 `gameover()` 游戏结束 togglePause()游戏暂停 等方法

## 游戏主窗口细节说明

沿用上述状态机的概念，采用枚举方式列出了游戏的状态(`enum game_state{READY, RUNNING, PAUSE, GAME_OVER};` 分别是重置、运行、暂停、游戏结束四个状态)，然后通过 `advance()` 方法根据游戏所处状态进行游戏画面刷新。

同时调用所有位于画布上的对像进行各部分的状态刷新

```
void Game::advance()
{
    // do nothing if game is not running 如果游戏未开始不做任何处理
    if(cur_state != RUNNING)
        return;

    // if mario is dead, game over 如果马里奥处于死亡状态，则游戏结束
    if(mario->isDead())
        gameover();

    // if mario is dying or transforming, animate/advance only him
    // (the rest of the world is freezed)  `the rest of-剩下的部分`
    // 如果马里奥处于正在死亡状态或者转化状态则只刷新 马里奥的状态
    // 其他对象会被冻结
    if(mario->isDying() || mario->isTransforming())
    {
        mario->animate();
        mario->advance();
        return;
    }

    //当马里奥进入管道时
    if(mario->ispiped()){...}

    //当游戏成功时
    ////////////////////////////////////////
    if(mario->isSuccess()){...}
    ////////////////////////////////////////
```

```
    // tell all game objects to animate and advance in the scene
    // 访问画布上的所有对象调用他们的animate()以及advance()方法
    for(auto & item : scene->items())
    {
        Object* obj = dynamic_cast<Object*>(item);
        if(obj)
        {
            obj->animate();
            obj->advance();

            // destroy died Entity objects
            // 清除已经失效的对象
            Entity* entity_obj = dynamic_cast<Entity*>(obj);
            if(entity_obj && entity_obj->isDead())
            {
                /*rintf("%s (%.0f,%.0f) destroyed\n", entity_obj-
>name().c_str(), entity_obj->pos().x(), entity_obj->pos().y());*/
                scene->removeItem(entity_obj);
                delete entity_obj;
            }
        }
    }

    // center view on Mario
    // 将摄像头移动到与马里奥附近
    centerOn(mario);
}
```

## 碰撞检测模块

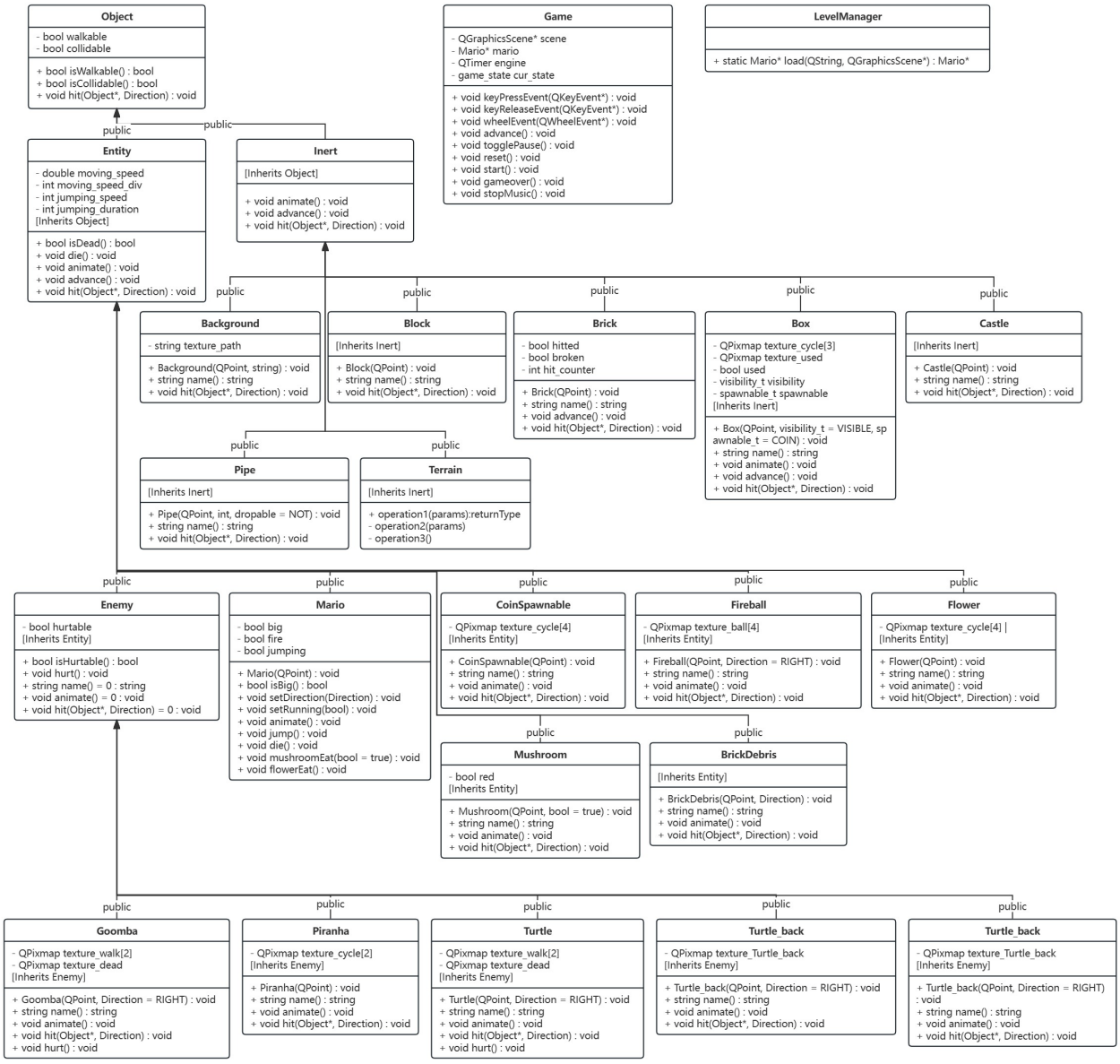
先将游戏中所有对象分为可移动对象(`Entity`)以及不可移动对象(`Inert`)，考虑到只有移动对象会与其他对象发生碰撞，所以我们在所有可移动对象父类(`Entity`)中实例化了QT内置方法`solveCollisions()`碰撞处理，通过与其子类及不可移动对象子类的`hit()`方法有机结合，写出了碰撞检测逻辑。如下:

```
void Entity::solveCollisions()
{
    // if the entity cannot collide or is 100% dead, we avoid
    // solving possible collisions
    // 解决bug 当对象处于死亡状态时没有能与之碰撞的物体
    if(!collidable || dead)
```

```
        return;

    // get collisions
    // 获取所有产生碰撞的对象
    QList<QGraphicsItem*> colliding_items = collidingItems();

    // will be set to true if we collide an impenetrable object
    // 当碰到障碍物时会被设置为true
    bool revert = false;

    // manage collisions
    // 处理所有产生碰撞的对象
    for(auto & ci : colliding_items)
    {
        // convert to game object, and skip if conversion
        // does not work (should never happen)
        // 处理碰撞队列为空的情况
        Object *obj = dynamic_cast<Object*>(ci);
        if(!obj)
            continue;

        // ignore collision if obj is not collidable
        // 忽略无法发生碰撞的物体
        if( ! obj->isCollidable())
            continue;

        // ignore collisions if a phantom Entity is involved
        // 忽略已经无效的怪物
        Entity* entity_obj = dynamic_cast<Entity*>(obj);
        if(entity_obj && (entity_obj->isPhantom() || phantom))
            continue;

        // ignore collisions between Mario's collectables and
Enemies
        // bug处理 忽略玛丽奥的收集物以及敌人 (当马里奥和敌人发生碰撞时调用敌人
的hit方法而非马里奥的)
        if( (dynamic_cast<Enemy*>(obj) && this->isCollectable()) ||
            (entity_obj && entity_obj->isCollectable() &&
dynamic_cast<Enemy*>(this)))
            continue;

        // get collision direction (获得收集物 当收集物与马里奥产生碰撞时)
```

```
        Direction coll_dir = collisionDirection(obj);

        // if it is not possible to calculate it, we skip current
collision
        // (e.g. this may happen if we have solved the collision
earlier)
        // bug处理 处理一些无法收集的物品，
        // 如我们已经将该收集物添加给马里奥但物品没消失的情况
        // 直接跳过该收集物
        if(!coll_dir)
            continue;

        // special case 1: touching a walkable object while falling
        // ---> end falling and set walkable object
        // 特殊情况1 当掉落的状态下碰到可移动对象时
        if(coll_dir == DOWN && falling && obj->isWalkable())
        {
            falling = false;
            walkable_object = obj;
        }

        // special case 2: touching an object while jumping
        // --> end jumping
        // 特殊情况二 处于跳跃过程碰到障碍物时：停止跳跃
        if(coll_dir == UP && jumping)
            endJumping();

        // the two objects hit each other
        // 当两个对象发生碰撞时分别调用两者的hit()方法
        obj->hit(this, inverse(coll_dir));
        this->hit(obj, coll_dir);

        // if we ended up here, it means we have to revert
        // to the previous position (impenetrable object)
        // 如果在这里结束说明我们需要刷新该对象的状态了(已经发生了碰撞)
        revert = true;
    }

    // revert to previous position if needed
    if(revert)
        setPos(prevPos);
}
```

# 整体类图




# 大小及性能

- 游戏大小大约占用100MB

- 系统运行占用CPU，系统内存少，运行流畅