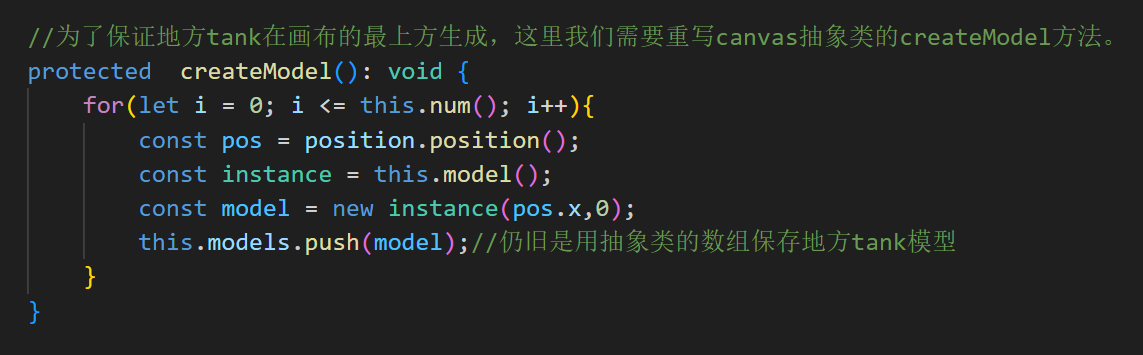
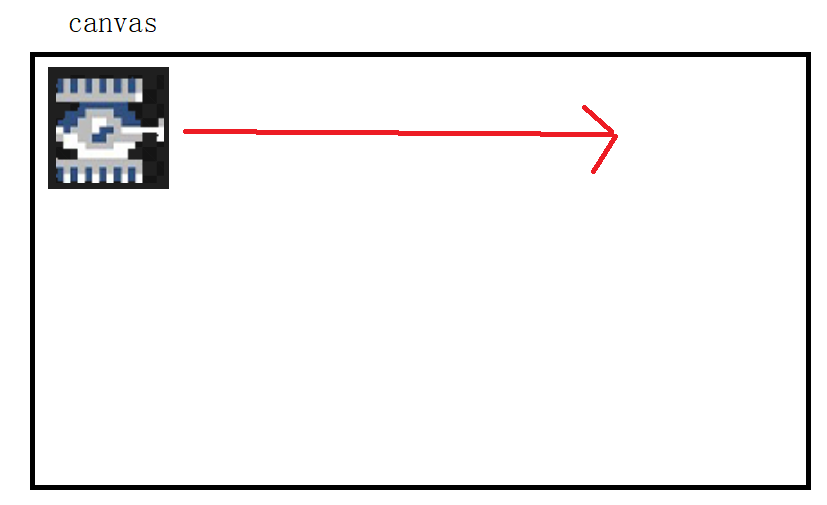
绘制敌方tank

首先我们应考虑如何让敌方tank出现在画布的上半部分。想让敌方tank只出现在画布的上半部分其实非常简单，我们可以统一将tank元素的初始坐标的y值设置为0，而x值则不变即可。所以这里我们要在tank的canvasClass中重写抽象类的createModel()方法。

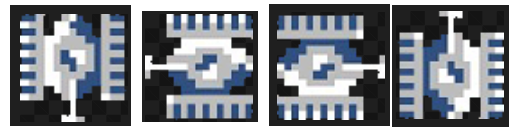


如何让坦克动起来？

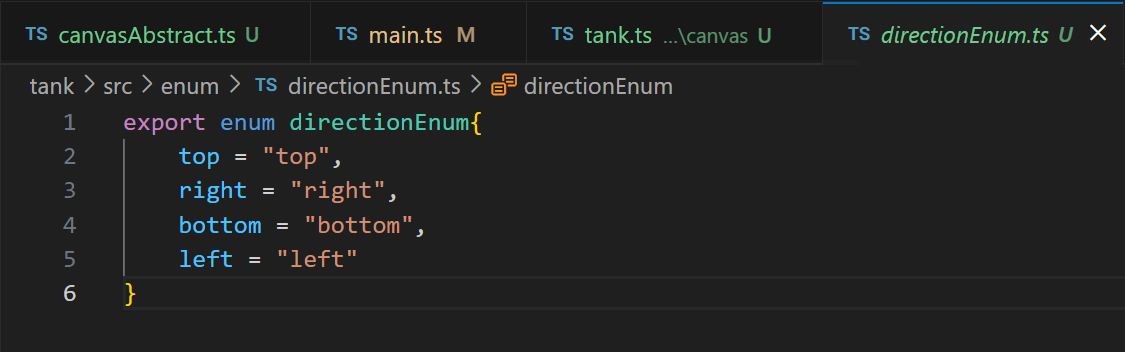
相让tank动起来其实也非常简单，你只需要不断的改变模型的坐标，然后不断的擦除画布然后重新绘制即可。



但我们tank的移动方向应该是四个方向，tank炮口的朝向随这坦克的前进方向改变而改变。所以我们应该准备好4个方向的tank图片。但考虑到后续我方坦克仍旧也是有4个移动方向，我定义一个enum类型的对象，对象中就是tank四个方向的字符串值（这个枚举类型的数据，在后面绘制我方tank的时候还会用到，所以就把它放到model的抽象类中吧）。



四个方向的tank图片。

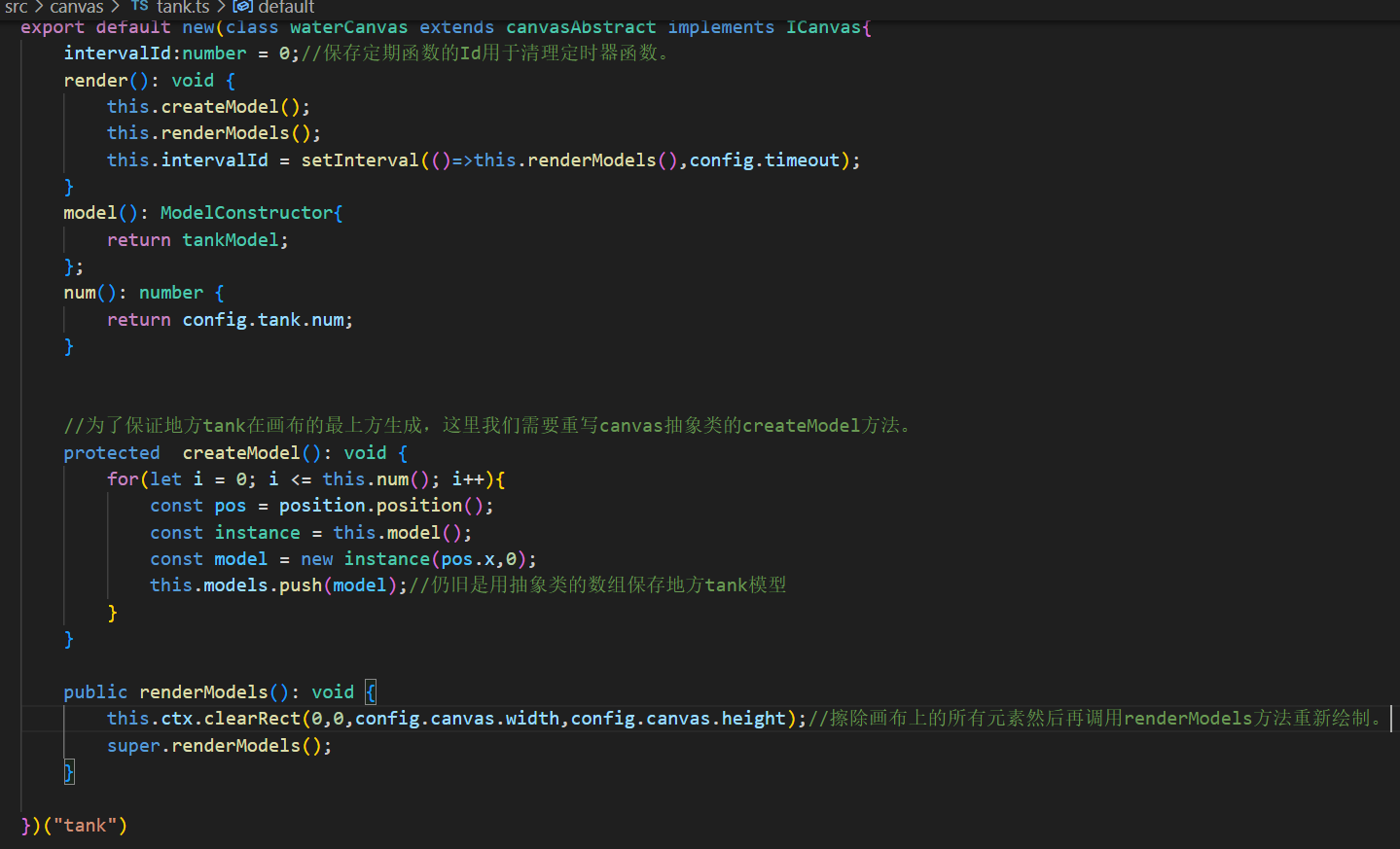


枚举类型

现在我们先默认这个tank的初始方向是炮口朝下。而实现tank移动的逻辑其实非常简单，我们在tank类中定义一个move方法用于控制坦克的移动。在这个方法中使while循环，让模型的x,y坐标随着坦克炮头的方向增减即可。利用switch条件判断，依据坦克炮头的方向增减对应模型在画布上的x，y坐标。最后在tank的canvas类中，使用定时器函数不断地调用render()方法，就可以实现坦克的移动。



实现tank移动的move方法。



Tank的画布类

为什么要保存定时器函数的Id

这里提一下，为什么要设置一个定时器Id：在Js中你创建的每一个定时器都存在一个定时器Id与之内部引用。什么意思呢？我们知道在Js中普通函数在执行完之后，由这个函数开辟的作用域空间也会随之消失，作用域空间其实就是内存上的一片空间。这是Js自动在帮我们做内存优化。但是定时器函数则不一样，由于定时器函数与定时器Id存在内部引用的原因，这个定时器函数在执行完的时候由它开辟的作用域空间并不会自动消失，所以Mdn网站的手册上给我们提供了一个clearInterval()方法用于清理定时器函数开辟的内存空间。



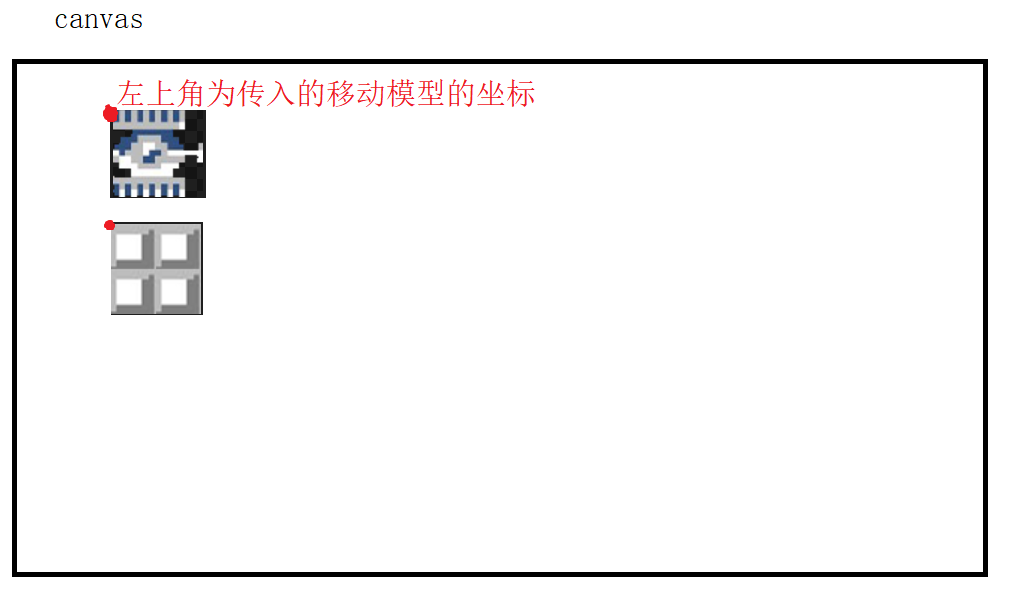
图示定时器函数之即使是在函数作用域内创建的定时器函数，它也不会消失。



图示定时器函数之即使你已手动清理了定时器，但是定时器Id还是会存在，说明它们之间存在内部引用。

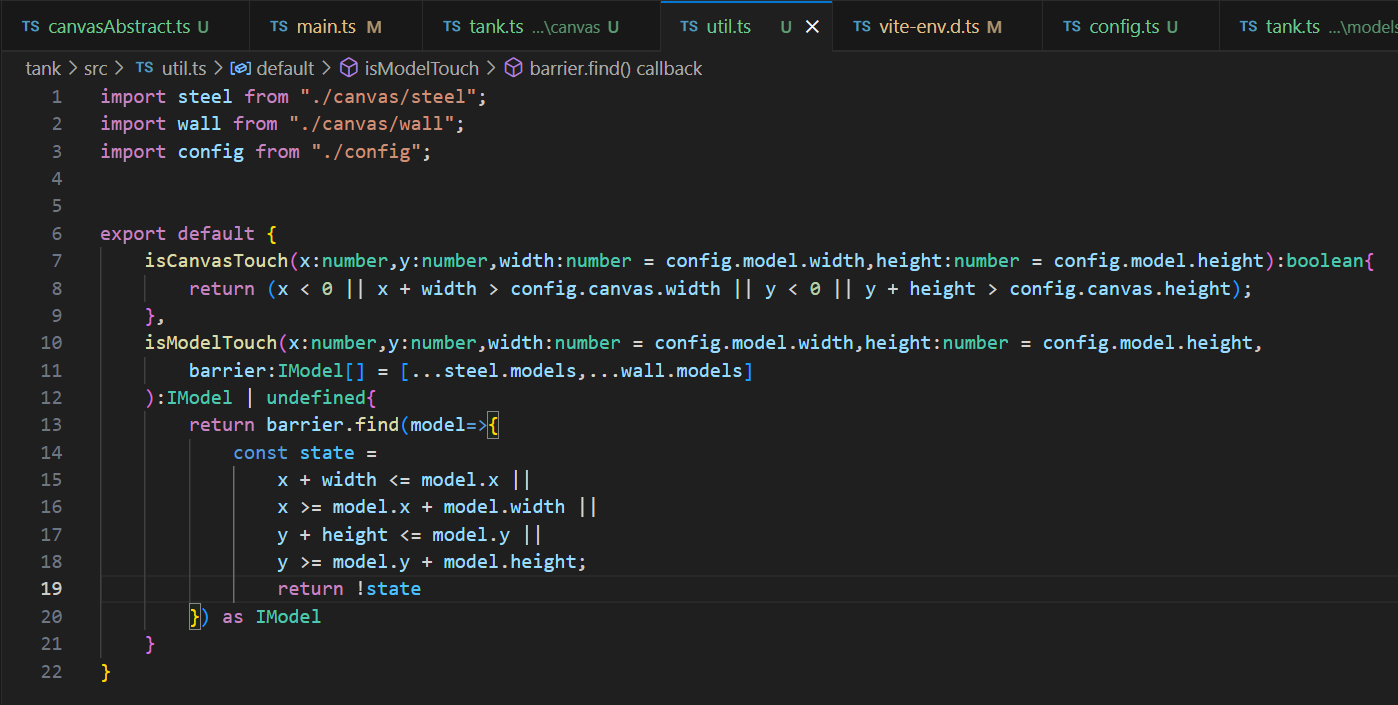
碰撞检测的原理是什么？

我们知道在写模型类的时候，我们给每一个模型的实例上都绑定了其在画布上的坐标。同时我们在canvas的抽象类中设置了一个IModel数组，用于保存在某一类元素在画布上绘制的模型的实例。所以我们可以在坦克正式改变坐标时，将改变了的坐标传入一个用于检测模型是否发生碰撞的函数中。这个函数接收的参数是可移动模型的x,y坐标，和模型的宽高。在方法中，我们将白墙，砖墙的的模型实例都压入到一个数组中。然后用数组的find方法去判断传入的移动元素的x,y坐标是否与画布上的白墙或者砖墙的模型碰撞。



具体的逻辑是，只要可移动模型的x坐标加上模型的宽度小于白墙或砖墙模型x坐标或者x坐标大于白墙或砖墙模型的x坐标加上模型宽度（y坐标同理），我就允许你继续移动，否则就代表你不能再移动了，需要改变方向。

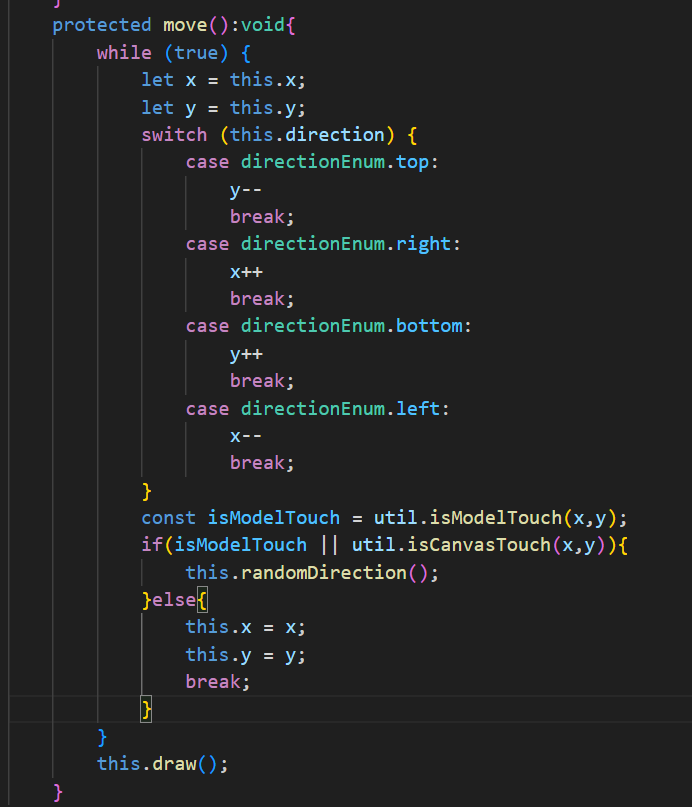
同时我们需要考虑当tank触碰到画布的边缘的时候，但这个比较简单。这里我们就不详细讨论了。



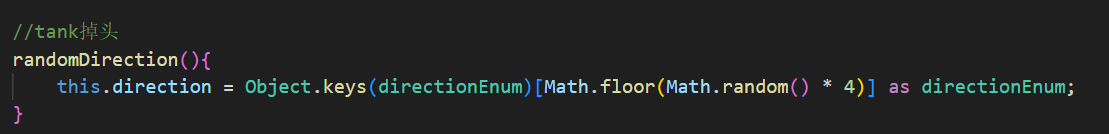
专门写一个utils文件向外抛出两个用于碰撞检测的方法，因为后续子弹和我方tank都要用到。

如何改变敌方tank的移动方向。

前面我们说过，当敌方坦克碰撞到无法穿过的模型的时候。我们需要让坦克的朝向更改，这个朝向的更改应该是随机的，且应该在tank模型的move方法中更改。所以我们需要一个当tank碰撞到某些元素时调用它，它能给我们返回一个随机的方向的方法。这个方法你可以写在模型类中，也可以写在抽象类中，但这里我们考虑到代码的可读性和简洁性还是将它放到模型的抽象类中。



敌方tank随机移动的算法。



随机改变tank方向的方法。