

[设计原因 2](#_Toc344473363)

[游戏概要 3](#_Toc344473364)

[游戏截图 3](#_Toc344473365)

[使用模块 4](#_Toc344473366)

[程序架构 5](#_Toc344473367)

# 设计原因

这次的作业发挥空间很大，我们也想过要不要写一个实用性较强的程序，如写一个问卷调查结果的统计程序，这样子的程序结构比较简单，也较容易实现，但在几番挣扎之后，我们最后决定向难度挑战，写一个游戏。不过因为Python中能导入的图片只有PMM和GIF格式，所以对于图片背景的处理有难度，所以我们便放弃了炫丽的画面，从而走向了8-bit的风格，而8-bit游戏经典中的经典就是太空侵略者(Space Invaders)了，虽然作为90后的同学可能对这个游戏不太熟识，但我相信大家对由像素组成的那个可爱的侵略者的身影和游戏的接口也会有点印象，而作为游戏迷的我们把这个经典之作再现则是别有一番感动！

# 游戏概要

太空侵略者在画面中左右移动，并随机放出攻击，每间隔一段时间太空侵略者会向下移动，而玩家需要避开攻击而且在太空侵略者落下前把他们击落。

# 游戏截图

|  |  |
| --- | --- |
| ScreenShot_0.jpg | HTP_2.jpg |
| ScreenShot_1.jpg | ScreenShot_2.jpg |
| ScreenShot_3.jpg | ScreenShot_4.jpg |

# 使用模块

我们的程序共导入了四个模块，分别是Tkinter，time，random，winsound，其中Tkinter是用于图像的导入和处理和按键的处理，time用于获取系统时间和使程序运行延时，random用于获取随机数，winsound用于导入声音。

# 程序架构

## 物件的构建

玩家面对的敌人为太空侵略者所组成的7\*5方阵，由于需判断攻击是否命中等，所以敌人、攻击和玩家都被设成独立的对象，都有对应的x，y坐标及其他状态值(如是否被击中等)，由于涉及数据的量比较多，所以矩阵方式来储存数据，以x=[[I for i in range(column)] for j in range(row)]的方式构造矩阵，再由for循环来赋与初值，为侵略的开始作好准备。

## 循环结构

程序中先定义函数rush()，再由Tkinter中的cv.after(10,rush)使rush函数中对自己再调用，形成递归来进行循环，由于此循环中rush()函数若附值会出错，也因数据量较多，若建立class的话需要在开始时把大量数据代入class中，容易造成混乱，所以则采用全局变量和函数的方式来取代class的使用。循环的使用成为了侵略战争的导火线，大战一触即发！

## 太空侵略者的侵略

游戏中的太空侵略者除了移动外，每隔一段时间还会有动作上的变化，而这个则是利用time中的time()函数来获取系统时间加以实现，如透过int(time())%2==0的语句可达至每隔1秒变换状态的操作，而且因为太空侵略者有动作上的变化，所以我们没有使用Tkinter中对象的move方法，而采用了刷新的方来进行动作的变化和移动，侵略者向我们步步进逼！

## Attack

太空侵略者的攻击则是采用一个矩阵来存放每个侵略者有否作出攻击的状态值，透过随机数来决定是否作出攻击，而每发攻击都有对应的坐标值，用于判断是否命中。因为敌人的移动较慢，为了加强游戏的刺激性，所以玩家每次只能发一颗子弹。透过各对象的对应坐标和尺寸，则可计算出物体是否有被击中，地球侵略战争全面爆发！

## 地球防卫战

操作按键是用了Tkinter中的bind功能，能够在按指定按键后执行对应的函数，若利用这些函数来修改全局变量，便能达到控制主程序的作用，如游戏开始时，玩家的坐标pX=360，也能透过键盘左、右键来控制玩家的X坐标，达到移动效果，按空格键来改变是否有发子弹的状态值等，藉着玩家的手来击落外层空间的侵略者！

## 路径的修改

因为程序中需要导入的图片比较多，考虑到玩家的使用和程序员的开发问题，所以我们在main()函数中设置了一个名为Path的全局变量，透过把这个变量修改成文件夹所在的位置，便可快速地修改导入图片的路径，而不用一行一行地修改，从而增加了便利性。