# 项目报告

## 主要功能

模仿保卫萝卜，我们实现了一个塔防游戏。

游戏中一共有三个地图供玩家选择，玩家可以使用change level按钮更换地图，不同的地图代表不同的难度。

游戏中一共分为三种怪物，这三种怪物的移动速度和血量均有不同，移速快的血量低，移速慢的血量高。这些怪物分波次从地图的一端逃逸到另一端，随着波数的增加怪物的血量也会逐渐提高。当逃逸成功的怪物数量到达一定数目时，游戏结束。

玩家可选择pillbox、wall、mines和artillery这四种防御措施来阻止怪物逃逸，其中pillbox指一个发射子弹的普通防御塔，wall是一个栅栏可以改变怪物的移动轨迹，mines是指地雷可以放置在怪物经过的路径上，用于炸毁怪物，artillery是一个巨型炮台，伤害比pillbox高但射速慢。

每消灭一个怪物，玩家将会获得相应的分数和金币，金币可用于购买上述的四种防御措施。玩家获得的分数也与玩家的游戏时间有关，随着游戏时间的增长，分数也会相应增加。

本塔防游戏配有背景音乐，背景音乐是使用python的midi库编写的。

## 实现原理

1. 地图加载

地图的加载由Level模块、Collision模块和Prefab模块实现。在Collision模块中，我们将屏幕上的点转化为图块的索引，在Level模块中，地图的基本信息，即各个图块的坐标，被存储在.level文件中，绘制地图时，加载.level文件，根据文件中的信息，为每个图块生成一个Prefab实例，Prefab实例会将该图块绘制在屏幕上。

1. 关卡加载

关卡的加载由Level模块、Collision模块、Prefab模块和Pathfinding模块实现。加载关卡时，首先将地图加载到屏幕上，每绘制一个图块就将该图块所在的矩阵设置为阻塞态，这用于碰撞检测。接着生成Pathfinding模块的一个实例，用于预先加载怪物的运动路径，最后设置关卡的基本数据，例如，时间、初始金币和生命值等。

1. 怪物分类

怪物共分为三类，如图所示，，从左至右怪物的血量依次提高，但移动速度依次降低。

怪物是由Enemy模块实现的，Enemy模块继承于Prefab模块，从.prefab文件中加载怪物的基本属性。Enemy模块调用Pathfinding模块生成怪物的运动路径，Enemy模块中还有几个函数用于实现怪物向目标移动、怪物到达目标后被销毁和受到伤害后的生命值变化等功能。

1. 防御类型

防御一共分为四种，如图所示，，从左至右名称依次为pillbox、artillety、mine和wall，其中pillbox的射速较快但伤害较低，artillety的射速较慢但伤害较高，mine用于埋伏敌人而wall用于改变敌人的运动路径。这些防御措施由Defence模块实现，Defence模块继承于Prefab模块，从.prefab文件中加载防御的基本属性。Defence模块中实现了查找目标函数，该函数是通过判断炮台和怪物的坐标关系来查找射击目标。这个模块还实现了转动炮台以跟踪目标的功能，通过调用其它模块，实现了pillbox的子弹攻击效果，artillety和mine的爆炸效果。

1. 具体操作

游戏开始后，玩家可以在怪物运动的路径上放置pillbox和aritllery，用于射击怪物，玩家也可以放置mine来埋伏敌人，每当怪物碰到mine时，mine会发生爆炸以此来伤害敌人。玩家还可以放置wall来阻止敌人的前进或改变敌人的运动路线。

1. 算法简介

怪物的寻路算法：怪物的运动路径是即时生成的，它和玩家放置防御措施的位置有关，每当怪物移动一步后，就会检测周围是否存在防御措施，如果存在，那么怪物将选择使玩家得分最低的路径，如果不存在，那么怪物将选择到达目的地的最短路径。我们还考虑到在怪物的运动过程中，玩家可能会放置防御措施来阻止怪物的移动，所以我们实现了一个repair函数，用于修改和再生路径。

## 主要模块

本项目中的主要模块有Prefab模块、Bullet模块、Collision模块、Defence模块、DefenceFlash模块、Enemy模块、Explosion模块、Game模块、Level模块、music模块、Pathfinding模块、Path模块、Wave模块和Window模块。

1. Prefab模块

Prefab模块是一个继承pygame的Sprite类的基础模块，Bullet模块、Defence模块、DefenceFlash模块、Enemy模块和Explosion模块均继承Prefab模块。Prefab模块的作用是从prefabs文件夹中的.prefab文件中加载各种配置。.prefab文件是文本文件，其中#开头的一行文字将作为注释而不做处理，其他行则由<变量名> : <变量类型> : <值>组成，从Prefab继承的模块类在其构造函数中指定.prefab文件，然后加载该文件并将变量添加到新的类实例中。

Prefab模块中定义了一个名为Cache的字典，用于存储加载的配置数据，这样就只需读取文件一次，每当需要使用配置数据时可从Cache中读取。

Prefab模块中主要的函数有update\_animation(delta)、load\_config(name)和apply\_config(config)。其中update\_animation(delta) 用于更新prefab上的所有sprite图表动画，delta是指从上一帧开始的时间；load\_config(name)用于获取指定prefab的配置字典，name是指要加载的.prefab文件的名称，该函数返回配置数据组成的字典。apply\_config(config)用于将所有的配置设置应用于prefab实例，config是指将要应用的变量值列表。

1. Bullet模块

Bullet模块继承于Prefab模块，主要功能是实现一个直线运动的子弹对象。Bullet对象在构造时接受两个（int，int）类型的坐标值以及一个int类型的伤害值。Bullet内建一个update函数用于在每一帧更新子弹的位置和状态。update函数中包括了触发到场景边缘的子弹销毁以及击中目标的子弹销毁和伤害计算，

1. Collision模块

Collision模块用于处理放置炮塔以及炮塔攻击时炮塔与网格的交互，同时也提供小怪前进的导航。根据游戏设定，我们需要避免炮塔堵死所有的前进路径，因此模块中提供point\_blocked以及rect\_blocked两个函数来判断一个坐标点或是一个矩形区域是否被阻塞。block\_point / unblock\_point / block\_rect / unblock\_rect则用于阻塞一个点或是一个矩形区域。由于游戏中有着点坐标与网格划分的矩形区域坐标两种索引方式，该类提供了一个point\_to\_index函数， 用于点坐标到矩形区域索引的转换。

1. Defence模块

Defence模块继承自Prefab模块，作为游戏中炮塔的基类。该类负责实现炮塔的索敌与攻击功能，并且更新炮台的图像。get\_target函数用于找到合适目标，is\_target\_suitable用于判断目标是否可以被攻击，update函数则用于每帧更新炮台的状态。

1. DefenceFlash模块

DefenceFlash模块继承自Prefab模块，用于绘制炮塔攻击时的动画效果。提供一个update函数用于逐帧更新图像。

1. Enemy模块

Enemy模块继承自Prefab模块，作为游戏中生成的敌人的基类。作为塔防游戏，敌军单位在场内的主要活动就是寻路并前进，Enemy模块中的update\_position方法即提供了这一功能。敌人碰触到目标时的交互则由reached\_target函数处理。敌人单位受到伤害以及被销毁的过程则由take\_damage与kill函数实现。

1. Explosion模块

Explosion模块用于绘制各种爆炸效果。和DefenceFlash类似属于图像效果的类，同样有一个用于实现逐帧更新图像的update函数。

1. Game模块

Game模块是游戏的主模块。该模块包含游戏循环和游戏的主要控制代码。load\_level函数用于加载关卡并初始化游戏相关类，run函数中包含了游戏的主循环过程，quit函数用于退出并关闭游戏。此外该类中还包含两个防御塔相关的函数，select\_defence函数用于给出玩家选择的防御塔类型，place\_defence则接受这一类型并在位置和费用均合法时放置防御塔。

1. Level模块

Level模块用于从.level文件中获取关卡信息，包括路径、碰撞数据、敌人的强度等等。Level模块包含三个函数，load\_data接受一个关卡的名称并加载对应的关卡数据，返回一个包含关卡数据的列表。start函数根据关卡数据设置好关卡的各项起始值并启动关卡。get\_score函数则包含了一个计算分数的公式。

1. music模块

music模块封装了pygame的一些音频接口，play\_music函数用于循环播放背景音乐，play函数则接受一个文件指针，用于播放特定的效果音。

1. Pathfinding模块

Pathfinding模块主要实现敌方单位的寻路。作为开放式地图塔防，敌人出发时可选的路径是不止一条的。Pathfinding模块会事先计算出多条路径并为每个敌人随机选择一条，当前路径被阻塞时，即切换到其它的路径。precompute函数接收一个整形变量，用于生成给定数量的路径。部分地图中，敌人的出发点不止一个，find\_start函数用于遍历找到所有的出发点。在路径生成后，get\_path函数用于给敌军单位选择一条路径。如果一个点被炮塔阻塞，repair函数会尝试通过改走旁边的格子来修复路径。而如果单位被新建的炮塔挡住，get\_partial\_path函数可以获取或者创建一条过指定点的路径，用于防止敌人被炮塔卡住。get\_point\_usage函数可以获得使用给定点的路径数量，用于get\_partial\_path与update函数中的路径重建。is\_critical函数可以判断一个点如果被阻塞是否会导致无法到达终点，如果是，会防止这个点被阻塞。

1. Path模块

Path模块内建了一个寻路算法用于寻找满足条件的指定路径。在路径被阻塞时，同样使用了repair函数来试图修复路径。start\_search函数中初始化了一些寻路所需的变量并开始寻路搜索，search函数则是封装了每次搜索使用的寻路算法。根据我们的设定，敌人的行为应倾向于使玩家尽可能少地获得分数，因此get\_lowest\_score函数可以查找玩家得分最低的路径。get\_neighbours函数可以获取指定位置的邻接图块，can\_use\_diagonal则判断两个坐标之间的对角线路线是否畅通，get\_cost则计算两个坐标之间的移动成本。以上三个函数均是在search函数中被调用用于辅助寻路的。

1. Wave模块

Wave模块控制每波敌人的生成。塔防游戏中敌人一般会一波一波地出现而非不断地出现，因此Wave模块内建了一个产生敌人的spawn函数并在每帧调用的update函数中构建了相应的逻辑来更新敌人的生成情况。enemy\_killed函数在敌人死亡时被调用，通过统计死去的敌人个数判断这波攻势是否结束。

1. Window模块

Window模块封装了pygame模块中的窗体对象。提供一个set\_title函数用于设置窗体标题、一个set\_background函数用于设置窗口背景色以及一个clear函数用于清空窗口。

## 音乐编写与播放

midi音乐对于玩游戏的同学一定不会陌生。记得小时候的很多flash游戏的背景音乐也是midi格式。所以有了给这个小游戏配上midi音乐的想法。

既然这门课程的核心内容是python语言的学习，我们队伍中的一名同学有了用python语言直接编写midi音乐的想法。上网查询发现，midi库提供了编写midi音乐的接口。但是midi库无法在windows上成功安装，所以我们在linux的虚拟机下进行了音乐的编写。

python里与midi音乐有关的库有十个左右，但是综合比较，midi库是可读性比较强的，操作也比较直接。虽然这个库在网上没有什么人写文档，库本身的文档也较为凌乱，但是通过不断的试探和实验还是可以分清楚每一个接口的作用的。

首先，如果你熟悉音乐制作和处理软件，会明白软件里以音轨为一个单位。midi库也是如此。它可以为一整段midi音乐添加音轨。为每一条音轨添加音符。它与python本身的语法有着很强的耦合——利用列表结构，一段midi音乐实际上是一个列表的嵌套。

首先，利用midi.Pattrtn()函数，建立一段midi音乐，然后，利用midi.Track()函数建立一条音轨，并append进midi音乐里。每条音轨本身也是一个列表结构，都可以append进音符数据。对于音符来说，midi.NoteOnEvent()接口相当于设置按下钢琴键的时刻，但这个时刻是一个相对时刻，即实际时刻是前面时刻的相加；midi.NoteOffEvent()接口是设置松开键的时刻，在这里设置时刻相当于这个音符的持续时值。对于这两个接口，分别有tick、velocity、pitch参数。其中tick即为时刻设置，velocity可以看做钢琴键的力度,pitch代表音高，在这里可以通过音名来设置，例如设置这个音为C5，就可以设置pitch=midi.C\_5。每条音轨完成后，需添加midi.EndOfTrackEvent(tick=1)的列表项作为音轨的结尾，否则电脑将无法播放这个文件。最后通过midi.write\_midifile()接口生成midi文件。只要接口使用得当，该midi文件是可以直接播放的。

在本项目中，因为是一个塔防游戏，所以想采用热血一点的音乐。于是数码宝贝的插曲《brave heart》列入了我们的考虑范围。考虑到一个一个音符非常繁琐，所以对这首歌进行了简化，钢琴左手配乐部分主要是八分音符打节奏，右手部分主旋律，一部分是和弦组成。由此共有4个音轨。全曲写完用了3000行左右。同时，生成的midi文件可以通过play\_music函数用于循环播放。

## 效果展示

开始菜单界面：



第一个地图的游戏界面：

****

第二个地图的游戏界面：



第三个地图的游戏界面:



游戏结束：

