* 1. pygame简介

pygame是一个功能强大的模组，可以管理图形、动画与声音，让基于Python的游戏开发更加便捷。开发者可以将工作重点放在程序逻辑上，而不必考虑各种繁琐的底层实现细节。在本章节中，我们将基于pygame模组，实现经典的手机游戏“贪吃蛇”。

* 1. pygame安装

在Windows操作系统中，pygame模组的安装并不复杂。在确保你的计算机已经安装了Python 3.x版本的情况下，打开命令行窗口，输入pip install pygame，pygame模组就可自动下载并安装。

* 1. 项目规划

开始开发之前，首先要对即将实现的程序有一个基本规划。我们期望实现的贪吃蛇游戏基本玩法是：玩家操控一条不断运动的蛇，得分方式是让蛇头经过在随机位置出现的豆子，使蛇“吃到”豆子，同时蛇身也会增长一个单位；随着得分不断提升，贪吃蛇也不断变长，玩家必须避免蛇头与身体相接触，否则游戏就宣告失败。游戏失败后，将会弹出提示信息，告知玩家最终得分，并重新开始游戏。

* 1. 开始开发

**1.4.1 创建pygame窗口，绘制游戏背景**

首先，我们新建一个python文件，并输入如下代码：

|  |
| --- |
| **import** pygame  **def** main():  width = 500  height = 500  rows = 20  win = pygame.display.set\_mode((width, height)) |

我们导入了pygame模组，我们将以此模组提供的功能为基础开发贪吃蛇游戏。接着我们定义了一个函数main()，该函数将作为游戏运行的主函数，负责绘制窗口、调用其他函数、响应玩家操作以定期更新游戏画面的全局性工作。我们调用pygame.display.set\_mode创建了一个名为win的窗口，游戏中的图形元素将在win中绘制。其中参数(width, height)是一个二元组，顾名思义，该二元组将设置游戏窗口的宽和高，单位为像素。在此，我们建立了一个宽和高均为500像素的正方形游戏窗口win。

由于贪吃蛇游戏实质上是由方格组成的“蛇”在窗口中运动并吃掉方格“豆子”的过程，所以游戏窗口本质也是被划分为方格组成的平面区域。为此，我们还定义了一个变量rows，用于规定按怎样的行（列）数将游戏窗口划分为小方格。该变量将会在后面用到。

接下来需要在主函数里添加一个循环语句块以更新游戏画面。因为在经典的贪吃蛇游戏中，蛇并非流畅移动，而是逐格移动的，具有一定的卡顿感。为了还原这种游戏体验，我们需要在每次更新游戏画面时，调用一些时间函数故意等待一小段时间，再绘制窗口。代码如下：

|  |
| --- |
| **import** pygame  **def** main():  --略—  flag = **True** clock = pygame.time.Clock()   **while** flag:  pygame.time.delay(50)  clock.tick(10) |

其中，pygame.time.delay(50)调用了pygame模组中的延迟执行函数，其参数为延迟执行语句的时间，单位为毫秒。同时，我们又定义了一个pygame内置的clock对象，并在循环内调用函数clock.tick(10)，含义是将游戏画面的更新速率控制在每秒10帧。这样，我们就可模拟出经典贪吃蛇游戏的“卡顿”观感。对于pygame.time.delay(50)和clock.tick(10)参数的设置，读者可自行调整以获得自己满意的游戏效果。

实现帧数控制后，就需要在循环中实际调用游戏画面绘制函数了。我们将编写一个函数redrawWindow()实现画面的重绘，并在循环体中调用该函数。

|  |
| --- |
| **import** pygame  **def** redrawWindow(surface):  surface.fill((0, 0, 0))  pygame.display.update()  **def** main():  -略-  *# 游戏主循环* **while** flag:  pygame.time.delay(50)  clock.tick(10)  redrawWindow(win)  main() |

函数redrawWindow()的参数是一个surface对象。在pygame中，surface是窗口的一部分，用于显示窗口中的元素。之前创建的win就是一个surface对象，通过display.set\_mode()返回的surface表示整个游戏窗口。在游戏画面更新循环开始后，通过执行redrawWindow，每次循环都将自动重绘该surface。

在主函数上方开始具体编写redrawWindow()函数。首先调用函数surface.fill()，其作用是设置窗口背景色，通过三元组参数可指定背景色编号。三元组的三个分量分别代表颜色中红色、绿色和蓝色的含量，取值范围为0-255。例如，(0, 0, 0)是黑色，(255, 0, 0)是红色，(255, 255, 255)是白色等。此处我们选用黑色作为背景色。然后调用pygame内置的display.update()函数对画面进行更新。

完成上述代码后，在中调用主函数main()，执行程序。你将会看到一个500\*500大小、黑色背景的游戏窗口。

接下来开始在窗口中绘制背景网格。首先定义两个全局变量rows和width，分别用于存储画面横向/纵向被划分为多少份，以及窗口的宽和高。为方便起见，我们将游戏窗口设定为正方形，因此画面的宽和高可只由一个变量width表示。通过这两个变量，我们可以控制构成贪吃蛇和和豆子的方格大小，并计算出每隔多少像素绘制一条网格背景线。

我们将绘制网格的操作放在函数drawGrid()中进行。该函数应在填充背景色之后、更新游戏画面之前被调用：

|  |
| --- |
| **import** pygame  **def** redrawWindow(surface):  surface.fill((0, 0, 0))  drawGrid(width, rows, surface)  pygame.display.update()  **def** drawGrid(width, rows, surface):  *'''绘制游戏背景方格'''* sizeBetween = width // rows  x = 0  y = 0   **for** l **in** range(rows):  x += sizeBetween  y += sizeBetween  *# 每个循环绘制一条纵向线和一条横向线* pygame.draw.line(surface, (255, 255, 255), (x, 0), (x, width))  pygame.draw.line(surface, (255, 255, 255), (0, y), (width, y))  **def** main():  -略-  main() |

drawGrid()函数有三个参数：游戏窗口的宽度width，方格行数rows，以及一个surface对象。通过width和rows相除，我们可以计算出背景中方格的边长大小，并将其存储在变量sizeBetween中。注意，由于方格边长单位为像素，所以必须为整数，因此这里使用运算符//表示结果向下取整。

接下来，定义两个变量x和y，它们将用于在绘制方格线时表示线条的起止点坐标。我们使用函数pygame.draw.line绘制方格线，它接受4个参数，依次为所绘制的surface对象、线条颜色、起点和终点坐标。对于每条横线而言，其起点与终点纵坐标恒定，绘制时用变量y表示起止点纵坐标，y每次递增一个方格的宽度以改变横线绘制位置；对每条竖线而言，其起点与终点横坐标恒定，绘制时用变量x表示起止点横坐标，x每次递增一个方格的宽度以改变竖线绘制位置。

完成上述代码后，执行，即可得到一个黑色背景、绘制白色方格线条的正方形窗口。

**1.4.2 编写方块类Cube**

无论是贪吃蛇还是它爱吃的豆子，其本质都是由一个或多个方块构成。因此需要编写一个方块类Cube对其进行一系列定义与操作。

首先，类似我们在主函数main()中做的那样，定义两个变量rows和width，存储窗口的像素宽度和窗口的方块行数。

**编写\_\_init\_\_()方法**

在Cube类的构造方法\_\_init\_\_()中，主要是定义四个变量并赋初值：位置坐标pos，赋初值start；表示水平和竖直移动方向的变量dirnx和dirny，分别赋初值1和0；表示颜色的三元组变量color。这些变量的初值均通过\_\_init\_\_()方法的参数获得。

其中，移动方向使用了两个变量dirnx和dirny表示。它们分别是directionX和directionY的缩写。这两个变量将有1，0或-1三种，用于表示贪吃蛇的移动方向。这里需要注意两点：

1. 对于pygame窗口而言，其坐标原点(0, 0)位于左上角，因此横坐标在右移时增大，左移时减小，纵坐标在下移时增大，上移时减小；
2. 第二，在贪吃蛇游戏中，只能进行水平和竖直方向移动而不能斜方向移动，因此dirnx和dirny两者必然有一个取0。如果这两个变量同时取非0值，则贪吃蛇的速度向量不与坐标轴平行，方块会沿着斜方向移动，这并不是我们想要的效果。

|  |
| --- |
| **import** pygame  **class** Cube(object):  width = 500  rows   **def** \_\_init\_\_(self, start, dirnx=1, dirny=0, color=(255, 0, 0)):  self.pos = start  self.dirnx = 1  self.dirny = 0  self.color = color  -略-  main() |

**编写move()方法**

对构成贪吃蛇的方块而言，它们的移动方向显然与贪吃蛇整体的移动方向相同。因此，将贪吃蛇的移动方向dirnx和dirny传入move()方法，即可表示方块的移动方向。之后，再调用self.pos()方法，结合原始位置与移动方向，更新方块位置，即可实现方块的移动。这里需要注意，在pos方法中，方块的坐标并不是pygame意义上的像素坐标，而是指“这是第几排第几列的方块”。

|  |
| --- |
| **import** pygame  **class** Cube(object):  width = 500  rows   **def** \_\_init\_\_(self, start, dirnx=1, dirny=0, color=(255, 0, 0)):  self.pos = start  self.dirnx = 1  self.dirny = 0  self.color = color   **def** move(self, dirnx, dirny):  self.dirnx = dirnx  self.dirny = dirny  self.pos = (self.pos[0]+self.dirnx, self.pos[1]+self.dirny)  -略-  main() |

**编写draw()方法**

Cube类的draw()方法除传递self参数外，还需要传递参数surface和一个布尔型变量eyes。Surface用于指定要在哪个surface对象上进行绘制，而变量eyes是为了区分当前要绘制的是头部方块还是身体其他部分。如果eyes取值为True，则在绘制时还将在方块上画出眼睛表示头部。

和绘制背景网格时类似，在draw()方法中也通过像素宽度width除以行数rows取得方格像素宽度dis。变量i和j则用于存储当前方格位置的横坐标和纵坐标（再次提醒，这里的“坐标”并不是像素坐标，而是指“位于第几排第几列的方格”）。

方块的绘制通过调用pygame的矩形绘制函数pygame.draw.rect()来实现。该函数的参数依次为：surface对象、绘制矩形的颜色、矩形的四个顶点的像素坐标。我们可以通过i、j与方格像素宽度之间的运算，推导出待绘制矩形的四个顶点坐标。注意，我们并未完全按照背景网格的尺寸绘制方格，而是稍微进行了缩小。这样使得背景网格线不至于被完全覆盖，一方面便于玩家对贪吃蛇长度和豆子的位置更容易把握，另一方面也能还原出老游戏的“像素感”。

最终，还需根据参数判断是否为头部，若为头部，则还需要执行if eyes下方的语句，画出贪吃蛇的眼睛。

|  |
| --- |
| **import** pygame  **class** Cube(object):  -略-   **def** draw(self, surface, eyes=**False**):  dis = self.width // self.rows  i = self.pos[0]  j = self.pos[1] pygame.draw.rect(surface, self.color, (i\*dis+1, j\*dis+1, dis-2, dis-2))  **if** eyes:  center = dis // 2  radius = 3  circleMiddle = (i\*dis+center-radius, j\*dis+8)  circleMiddle2 = (i\*dis+dis-radius\*2, j\*dis+8)  pygame.draw.circle(surface, (0, 0, 0), circleMiddle, radius)  pygame.draw.circle(surface, (0, 0, 0), circleMiddle2, radius)  -略-  main() |

**1.4.3 编写贪吃蛇类Snake**

**编写\_\_init\_\_()方法**

根据游戏规则，每成功吃下一颗豆子，贪吃蛇的身体长度都将加1，因此我们可以将一个贪吃蛇对象看作是由一组方块构成的对象。因此可以创建一个空列表body，用以存储构成躯干的方块。此外，还要创建一个字典turns来存储贪吃蛇躯干的拐点坐标及拐向。

下面编写Snake类的构造方法\_\_init\_\_()。除self参数外，\_\_init\_\_()方法有color和pos两个参数，分别用于传递贪吃蛇的颜色和头部位置起始坐标。在方法主体部分，首先要将参数的颜色保存在对应变量，并在指定位置pos新建一个Cube对象（稍后将对其进行编写），然后将头部方块添加到body列表中。

接下来定义两个变量dirnx和dirny，变量dirnx和dirny将用于贪吃蛇的移动方法move()中。Snake及其构造方法代码如下所示。

|  |
| --- |
| **import** pygame  **class** Cube(object):  -略-  **class** Snake(object):  body = []  turns = {}   **def** \_\_init\_\_(self, color, pos):  self.color = color  self.head = Cube(pos)  self.body.append(self.head)  self.dirnx = 0  self.dirny = 1  -略- |

**编写move()方法**

move()方法将根据不同事件作出不同的动作响应。在接受按键输入时，我们使用for循环配合pygame.event.get()配合来监听用户事件，而在循环体内部使用if语句对不同的事件作出针对性响应。首先要关注的是退出事件。当监听到事件类别为pygame.QUIT时，要执行退出游戏动作。

为了获得当前键盘所有按键的状态（按下/松开），可使用pygame.key.get\_pressed()函数。该函数将返回一组布尔值，表示键盘上每个按键的状态。该列表是使用按键常量进行索引的（例如pygame.K\_LEFT表示左方向键），如果取值为True则表示某按键被按下了。我们将pygame.key.get\_pressed()返回的结果存储在变量keys中。

随后，遍历列表keys，针对不同的按键状态，使用if语句作出不同响应。既然是move()方法，自然应该对四个方向键的输入有着相应动作。需要注意的是，在pygame中，原点（即坐标为(0, 0)的点）和数学中的坐标原点不同，并不位于窗口的中心位置，而是位于窗口左上角。因此执行向左、向右移动时，横坐标应分别减小、增大；执行向上、向下移动时，纵坐标也应分别减小、增大。同时还应注意，在贪吃蛇游戏中，只能向上下左右四个方向水平或竖直移动，而不能斜方向移动，因此dirnx和dirny必然不能同时为非零值。

根据上述规律，可以对应地写出上、下、左、右移动时，变量dirnx和dirny的变化情况。以向左移动为例，此时dirnx应取-1，dirny取0。因此分别给self.dirnx和self.dirny赋值-1和0。最后。将当前位置坐标作为键，长度为2的一维列表作为值（该列表的元素包含当前方向的dirnx与dirny），将该键-值对存放在字典turns中。其他方向的移动可以此类推。由此可得move()方法的代码如下所示:

|  |
| --- |
| **import** pygame  **class** Cube(object):  -略-  **class** Snake(object):  body = []  turns = {}   **def** \_\_init\_\_(self, color, pos):  -略-  **def** move(self):  **for** event **in** pygame.event.get():  **if** event.type == pygame.QUIT:  pygame.quit()   keys = pygame.key.get\_pressed()   **for** key **in** keys:  **if** keys[pygame.K\_LEFT]:  self.dirnx = -1  self.dirny = 0  self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]  **elif** keys[pygame.K\_RIGHT]:  self.dirnx = 1  self.dirny = 0  self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]  **elif** keys[pygame.K\_UP]:  self.dirnx = 0  self.dirny = -1  self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]  **elif** keys[pygame.K\_DOWN]:  self.dirnx = 0  self.dirny = 1  self.turns[self.head.pos[:]] = [self.dirnx, self.dirny] |

仅仅对按键作出响应还不够，还需要实现贪吃蛇的移动。由于贪吃蛇处于不停运动的状态，又由一系列方块构成其躯干，所以还需要遍历构成其躯干的方块，通过对它们的逐个移动实现贪吃蛇整体的移动。同时还应注意，贪吃蛇的躯干方块位置可能有两种情形：一种是普通点，另一种则是恰好位于躯干转向的拐点。对于后者，我们通过对比贪吃蛇躯干构成方块的位置坐标与字典turns，如果以该方块所在位置为键，在字典turns中查询到了键值对，则执行拐点的移动操作，否则执行普通点的移动操作：

|  |
| --- |
| **import** pygame  **class** Cube(object):  -略-  **class** Snake(object):  -略-   **def** move(self):  -略-  **for** i, c **in** enumerate(self.body):  pos = c.pos[:]  **if** pos **in** self.turns:  turn = self.turns[pos]  c.move(turn[0], turn[1])  **if** i == len(self.body) - 1:  self.turns.pop(pos)  **else**:  **if** c.dirnx == -1 **and** c.pos[0] <= 0:  c.pos = (c.rows-1, c.pos[1])  **elif** c.dirnx == 1 **and** c.pos[0] >= c.rows - 1:  c.pos = (0, c.pos[1])  **elif** c.dirny == 1 **and** c.pos[1] >= c.rows - 1:  c.pos = (c.pos[0], 0)  **elif** c.dirny == -1 **and** c.pos[1] <= 0:  c.pos = (c.pos[0], c.rows - 1)  **else**:  c.move(c.dirnx, c.dirny) |

在上述新增代码中，最外层是一个遍历所有贪吃蛇躯干方块的循环，其中i为索引变量，c为方块对象Cube的简写。首先，语句pos = c.pos[:]拷贝了当前方块的位置坐标；随后对该坐标pos在拐点字典turns中进行检查。

如果以坐pos标为键在字典turns中可以查到键值对，则说明该点是拐点。根据该键值对给定的移动方向（回忆前面的内容，这个移动方向用一个长度为2的一维列表表示，组成元素分别为表示水平向和竖直向的移动方向的两个整数），使用方块对象的move()方法移动方块；在移动完毕后还要判定，如果该方块是贪吃蛇躯干最后一个方块，则应将该拐点从turns字典中删除。

如果以坐标pos为键在字典turns中查询不到键值对，则说明该点是普通点。如果方块移动过程中遭遇了窗口边界，则根据当前方块的移动方向和边界方向不同，为方块的当前位置赋以不同的新值：

（1）当方块方向变量dirnx取-1（向左移动），且方块横坐标取值小于0（到达左边界）时，我们就将方块移动到屏幕右侧，保持其纵坐标不变，实现循环左移；

（2）当方块方向变量dirnx取1（向右移动），且方块横坐标取值大于c.rows-1（到达右边界）时，我们就将方块移动到屏幕左侧，保持其纵坐标不变，实现循环右移；

（3）当方块方向变量dirny取1（向下移动），且方块纵坐标取值大于c.rows-1（到达下边界）时，我们就将方块移动到屏幕上方，保持其横坐标不变，实现循环下移；

（4）当方块方向变量dirny取-1（向上移动），且方块纵坐标取值小于0（到达上边界）时，我们就将方块移动到屏幕下方，保持其横坐标不变，实现循环上移；

（5）若不属于上述四种到达边界的情形，则方块在窗口内按给定方向正常移动，即执行方法c.move(c.dirnx, c.dirny)。

**编写draw()方法**

通过上述工作，我们实现了贪吃蛇的移动逻辑，但还需要将贪吃蛇在窗口中绘制出来。对贪吃蛇的绘制本质上是对构成贪吃蛇的一系列Cube对象的绘制。因此对于Snake的draw()方法，实质就是遍历构成贪吃蛇的方块，调用Cube的draw()方法。这里唯一需要注意的一点是，为区分显示头部，我们需要对构成头部的方块进行区别绘制。

**1.4.4 更新主函数main()，检验Snake与Cube类编写情况**

以上基本完成了Snake和Cube两个类的编写，下面需要对主函数进行更新，以检查之前的代码是否能正常工作。

首先新建三个全局变量width、rows和s，分别用于存储游戏窗口像素宽度、游戏窗口网格行数和贪吃蛇对象。相比原来的代码，我们将width和rows全局化，更易于操作。随后，将s初始化为Snake对象，并通过参数设置其初始颜色为红色(255, 0, 0)，初始方格坐标为(10, 10)。在使用redrawWindow()函数重绘窗口之前，我们添加调用s.move()方法，在每次重绘窗口之前检测按键输入情况，并更新窗口中对象的位置，最后通过redrawWindow()函数将更新结果绘制出来。

|  |
| --- |
| **import** pygame  **-略-**  **def** main():  **global** width, rows, s  width = 500  rows = 20  win = pygame.display.set\_mode((width, width))  s = Snake((255, 0, 0), (10, 10))  flag = **True** clock = pygame.time.Clock()   # 游戏主循环**while** flag:  pygame.time.delay(50)  clock.tick(10)  s.move()  redrawWindow(win)  main() |

完成主函数main()的更新后，运行程序，可以看到一个只有头部、长度为1的红色小蛇（现在只是一个方块）默认向右方移动。按键盘方向键可以操作小蛇，改变其移动方向。

**1.4.4 实现豆子的随机位置生成与吃豆**

**编写豆子位置随机生成函数randomSnack()**

在确认之前的代码可以正常运行后，我们继续编写随机生成豆子坐标的函数randomSnack()。这个函数将在窗口的随机位置生成豆子供贪吃蛇食用，每吃掉一颗豆子，贪吃蛇的长度都会增加。注意，虽然豆子的位置时“随机生成”的，但也不是任意的，我们需要保证不能让豆子的生成位置与当前贪吃蛇的位置重合。

由于我们在随机确定位置时需要用到随机函数，因此在编写randomSnack()之前，首先要在文件头部用import random导入随机功能模组。

randomSnack()函数接受两个参数输入，依次是窗口方格行数rows和贪吃蛇对象snake。在函数主体部分，首先通过赋值语句positions = snake.body将贪吃蛇的躯干信息列表复制到变量positions中。随后，调用随机函数random.randrange()，生成横坐标和纵坐标，并存储于变量x和y中。接下来，通过filter函数过滤出贪吃蛇躯干方块所处的位置坐标。若生成的随机坐标(x,y)与贪吃蛇躯干坐标相同，则重新生成，直到(x,y)不再与贪吃蛇躯干重合为止。最终，randomSnack()返回表示豆子坐标的二元组(x,y)。

|  |
| --- |
| **import** pygame **import** random  -略-  **def** randomSnack(rows, snake): positions = snake.body   **while True**:  x = random.randrange(rows)  y = random.randrange(rows)  **if** len(list(filter(**lambda** z:z.pos == (x, y), positions))) > 0:  **continue  else**:  **break   return** (x, y)  **def** main():  -略-  main() |

**编写贪吃蛇增长方法addCube()**

贪吃蛇在吃到豆子后，身体将会变长。这个动作将在即将编写的方法addCube()中实现。这是一个属于Snake对象的方法。

贪吃蛇身体的增长——即躯干方块的增加——显然在尾部进行。我们通过语句tail = self.body[-1]找到当前贪吃蛇的尾部，并将当前贪吃蛇的运动方向存入变量dx和dy中。之后，我们通过一组if语句，根据贪吃蛇当前运动方向的不同，在不同位置添加新方块。例如，当贪吃蛇向左移动时，应在尾部方块的右侧添加新方块；当贪吃蛇向上移动时，应在尾部方块的下方添加新方块等。其他情形可以此类推。最后，给尾部方块的速度变量赋值，让新增的方块也能跟上前方“大部队”的步伐。

|  |
| --- |
| **class** Snake(object):  -略-  **def** addCube(self):  # 获得尾部方块的运动方向  tail = self.body[-1]  dx, dy = tail.dirnx, tail.dirny  # 根据当前运动方向，确定新方块添加位置  **if** dx == 1 **and** dy == 0:  self.body.append(Cube((tail.pos[0]-1, tail.pos[1])))  **elif** dx == -1 **and** dy == 0:  self.body.append(Cube((tail.pos[0]+1, tail.pos[1])))  **elif** dx == 0 **and** dy == 1:  self.body.append(Cube((tail.pos[0], tail.pos[1]-1)))  **elif** dx == 0 **and** dy == -1:  self.body.append(Cube((tail.pos[0], tail.pos[1]+1)))  # 为新方块赋值运动速度  self.body[-1].dirnx = dx  self.body[-1].dirny = dy |

**在主函数中实现吃豆效果**

完成上述功能后，我们还需要更新主函数main()去调用并实现吃豆效果。首先需要创建一个Cube对象作为豆子。创建时，将randomSnack(rows, s)作为参数，随机确定豆子的生成坐标，另一个参数为豆子颜色，通过一个三元组确定。

接下来编写吃豆后贪吃蛇增长的逻辑。在贪吃蛇头部触碰到豆子后，可判定为贪吃蛇成功“吃到了”豆子，贪吃蛇的长度也随之增加。

|  |
| --- |
| **def** main():  **global** width, rows, s, snack  width = 500  rows = 20  win = pygame.display.set\_mode((width, width))  s = Snake((255, 0, 0), (10, 10))  snack = Cube(randomSnack(rows, s), color=(255, 255, 255))  flag = **True** clock = pygame.time.Clock()   # 游戏主循环**while** flag:  pygame.time.delay(50)  clock.tick(10)  s.move()   # 检测是否吃到了豆子 **if** s.body[0].pos == snack.pos:  s.addCube()  snack = Cube(randomSnack(rows, s), color=(255, 255, 255))   redrawWindow(win) |

完成主函数的更新后，可重新运行Python程序，测试是否能正常实现吃豆-增长的功能。

**1.4.5 实现游戏结束与重启机制**

当贪吃蛇头部与自身躯干相撞时，游戏即宣告结束，此时将提示玩家game over、给出得分，并重启游戏。和吃豆一样，游戏结束的碰撞检测将在主函数main()中进行。进行主函数更新之前，需要注意，在文件头部增加rt tkinter as tk以及om tkinter import messagebox两行代码，以导入使用消息框所必须的模组。

进行碰撞检测时，将遍历贪吃蛇躯干的每一个方块，检测其后续方块是否有与其位置相重合者。若存在，则说明发生了碰撞，游戏宣告结束。同时，将前贪吃蛇的长度乘以10作为最终得分，显示在消息框内。

|  |
| --- |
| **import** pygame **import** random **import** tkinter **as** tk **from** tkinter **import** messagebox  **def** main():  -略-  # 游戏主循环  **while** flag:  pygame.time.delay(50)  clock.tick(10)  s.move()   # 检测是否吃到了豆子  **if** s.body[0].pos == snack.pos:  s.addCube()  snack = Cube(randomSnack(rows, s), color=(255, 255, 255))  # 游戏结束碰撞检测  **for** x **in** range(len(s.body)):  **if** s.body[x].pos **in** list(map(**lambda** z:z.pos, s.body[x+1:])):  messageBox(**'GAME OVER'**, **'Final score: %s'** % (str(10\*len(s.body))))  s.reset((10, 10))  **break** redrawWindow(win) |

完成主函数的更新后，再次重新运行游戏进行测试。此时，贪吃蛇应该能正常完成吃豆、增长、头身相接触后游戏结束、出game over提示与最终得分后游戏重启的功能。

#end