**贪吃蛇游戏实验报告**

叶欣 5120209375

1. **概述**

本游戏主要对象包括蛇、食物和墙，在游戏中通过鼠标控制蛇的前进方向，以实现它们的互动。蛇由双向队列数据结构实现，墙涉及到文件的读取，并且主要由二维列表实现，食物涉及到随机放置的问题。三个类的互动包括：1、食物的放置需要根据蛇与墙的位置决定，以避免重叠；2、蛇是否吃到食物需要对蛇类和食物类进行综合判断；3、蛇是否撞到墙需要对蛇类和墙类进行综合判断。主函数控制程序的流程，包括对话框和组件的创建，主循环中嵌套了游戏前设置循环、游戏循环等部分。

**二、类与数据结构的实现**

1、蛇类

该类包括以下简单变量：头的xy坐标、头的朝向、蛇的长度。头的坐标与朝向决定着吃食、自吃、撞墙等判断。每次蛇前进，头部坐标发生变化；每次鼠标控制转向，头部朝向发生变化。尤其注意当蛇出了游戏界面时，头部坐标有一个突变，使得蛇从游戏界面的另一端出现。蛇的长度决定着蛇的闪烁动画、蛇的擦除是否成功。

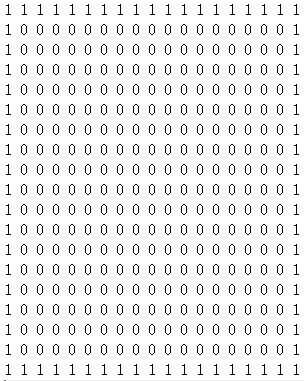
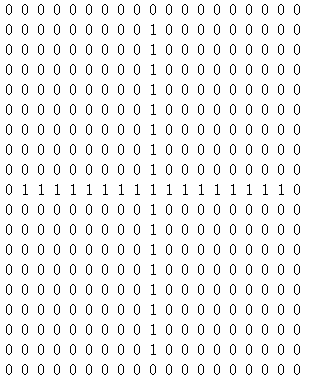
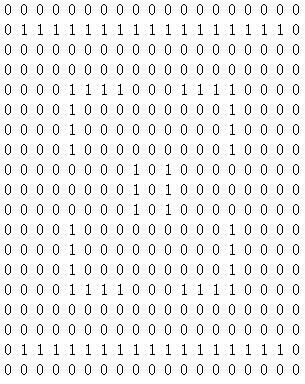
该类还包括以下复杂数据结构：蛇的每一节的坐标保存在2倍蛇长的规模的列表snake.q中，如图一所示，蛇的每一节的形状保存在长度与蛇长一致的列表snake.shape中。在检查蛇是否自吃时，需要将头部坐标与列表q中头部以后的坐标进行比对，如果有重复则发生自吃。在放置食物时也要将食物的新坐标与q比对，如果重复则必须重新设置食物位置。蛇每前进一步，在q开头添加新头部的坐标，如果这次前进有吃到食物，则尾部不做处理，蛇的净长度加一；如果没有吃到食物，则尾部去掉一组坐标，蛇的净长度不表。在对话框中画出蛇需要使用shape列表，内容为Rectangle类，以每一节的坐标为中心绘制方形，并设置颜色。每次移动，头部插入新的Rectangle类，如果这次移动没有吃食，尾部undraw并pop掉。在重新开始游戏时，原来的shape所有元素都要undraw清除，在蛇闪烁时，所有元素交替draw和undraw。



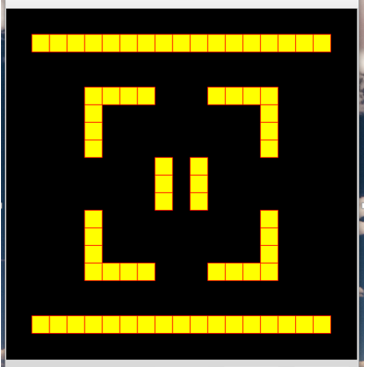
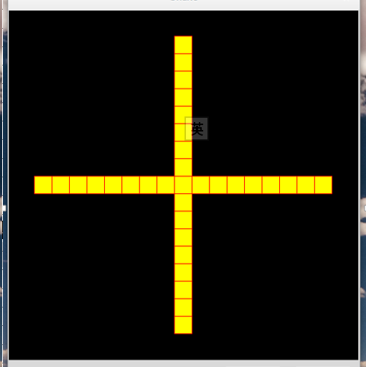
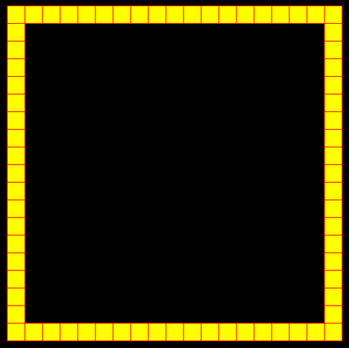
图一、初始化产生的snake.q

2、墙类

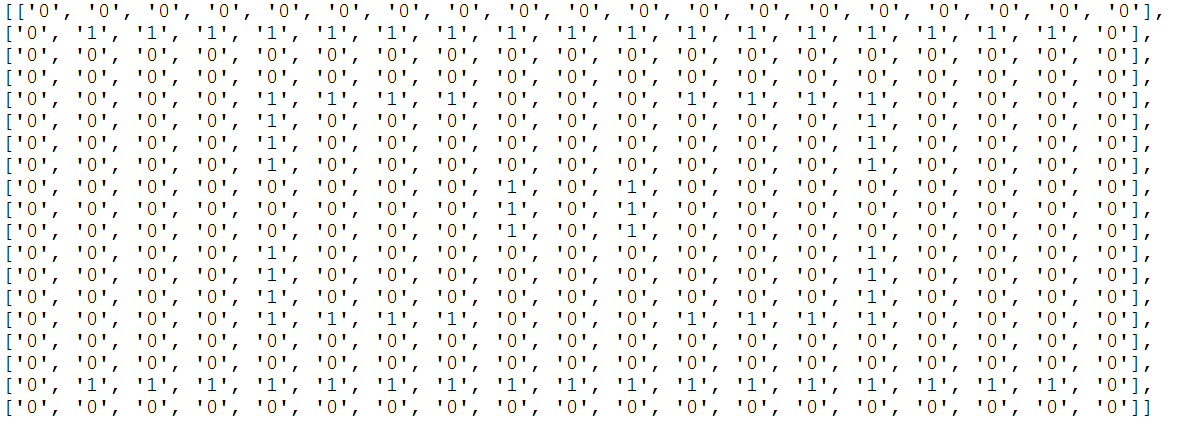
该类包括以下数据结构：包含游戏界面中所有单元格是否为墙的信息的19\*19二维列表linelist，包含所有墙单元的xy坐标的二维列表WallPos，包含每一个墙单元的形状Rectangle类的一维列表WallBlocks。这些列表的初始化通过文件读取完成，这些文件为txt文件，游戏界面中单元格有墙则对应位为1，否则为0，三个文件如图二所示，显示测试的截图如图三所示。通过readline()得到一行的字符串，再通过split()获得按空格拆分的列表，对每一行进行上述操作后append到linelist中，如图四所示。在根据linelist对应项是否为”1”生成WallPos，如图五所示，并绘制出WallBlocks的每一元素。在判断蛇是否撞墙时，只需查找头部坐标对应的linelist的元素是否为”1”，在放置食物时要将食物的新坐标与WallPos比对，如果重复则必须重新设置食物位置。在更改地图时，要将WallBlocks元素全部undraw()后再生成新的Wall实例。

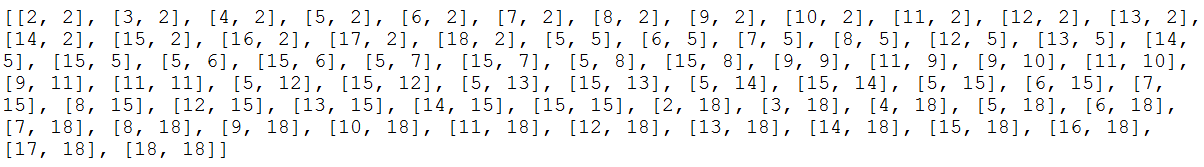
图二、Wall1.txt，Wall2.txt，Wall3.txt



图三、三种墙壁的效果



图四、Wall3.txt产生的wall.linelist



图五、Wall3.txt产生的wall.WallPos

3、食物类

食物类相对简单，变量包括xy坐标和可绘制的Circle类变量food。在判断是否吃到食物、放置食物是否与蛇或墙重叠时都要用到xy坐标。每次吃掉食物，都要对food进行move(dx,dy)操作。

1. **类的互动关系**

前文已述蛇、食物、墙之间的互动关系，整理如下：蛇是否吃到食物的对应、蛇是否撞墙的对应、食物放置是否蛇身体重合、食物放置是否与墙的位置重合。

蛇是否吃到食物的对应由函数CheckFoodEat(snk\_x,snk\_y,h,fd\_x,fd\_y)实现，参数包括蛇的头部坐标与方向、食物的坐标，返回值为1或0。是否吃到食物的判断必须在蛇到达食物位置之前一步进行，否则会出现有一步看不到食物位置的情况，另外还需注意蛇从游戏界面一侧出去，从另一侧进入吃食的情况。函数按照头部朝向分了4种情况进行判断，根据头部朝向，蛇和食物的xy坐标中有一组必须相等，另一组或提前一步，或差值为游戏界面边长，否则就是没有吃到食物。

蛇是否撞墙的对应由函数food.CheckIntoWall(self,wall)实现，该函数查找蛇头部坐标对应的墙的linelist的元素是否为”1”。如果是则撞墙。

食物放置由food.FoodPosGenerate(self,snake,wall)实现，在循环中产生游戏界面尺寸范围内的两组随机整数xy坐标，之后判断是否与旧坐标重复，再分别判断是否蛇身体重合、是否与墙的位置重合。如果任何一个发生重合，循环继续，重新生成随机坐标，如果没有重合，跳出循环，如果循环30次仍无法生成合适的食物位置，则放弃生成。

食物放置是否蛇身体重合由函数snake.FoodInterf(self,fd\_x,fd\_y)实现，是否与墙重合由函数wall.FoodInterf(self,fd\_x,fd\_y)实现。两者方法相同，分别尝试（try）对snake.q和wall.WallPos使用index([fd\_x,fd\_y])，如果找到相同项，不报错，则说明出现重合，如果找不到相同项，报错进入except，说明没有重合。两种情况分别返回1和0。

1. **游戏进程控制**

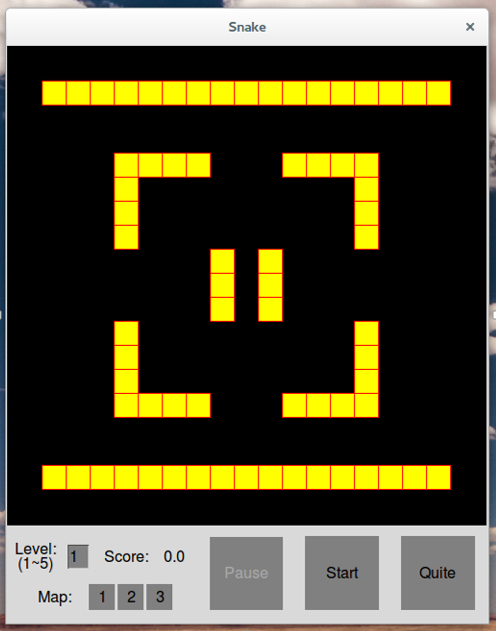
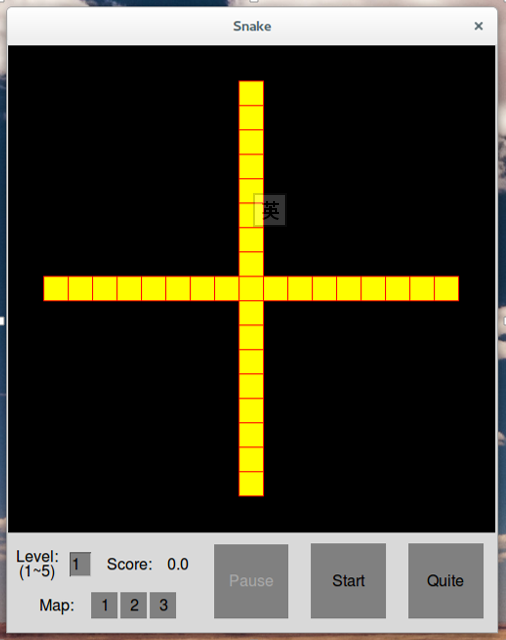
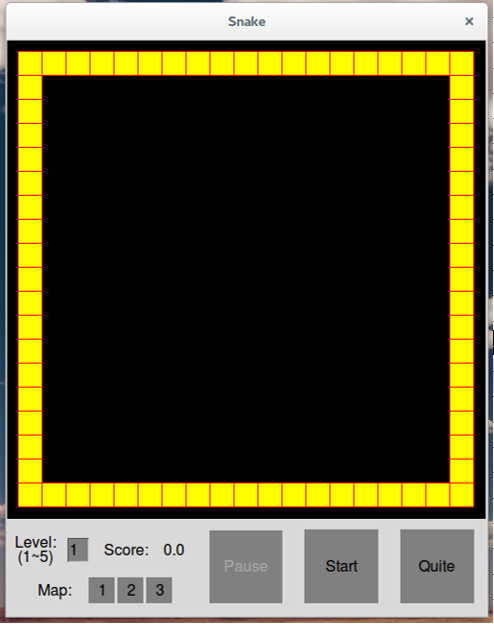
游戏控制由鼠标进行，由函数Click2Turn(clk\_x,clk\_y,head,snk\_x,snk\_y)处理。游戏进程主要由main()函数控制，考虑到游戏有多种GameOver条件，最终却有同样的结果，为提高代码重用性，使用GameOver (snake,food,Bt,w)函数统一处理。

鼠标控制蛇的转向方法如下：如果蛇头横向运动，则点击比蛇头位置高的地方向上转弯，点击比蛇头位置低的地方向下转弯；如果蛇头纵向运动，则点击比蛇头位置靠左的地方向左转弯，点击比蛇头位置靠右的地方向右转弯。Click2Turn(clk\_x,clk\_y,head,snk\_x,snk\_y)如上所述，先判断蛇运动是横向还是纵向，再根据鼠标点击位置坐标和蛇头坐标的大小关系决定新朝向。

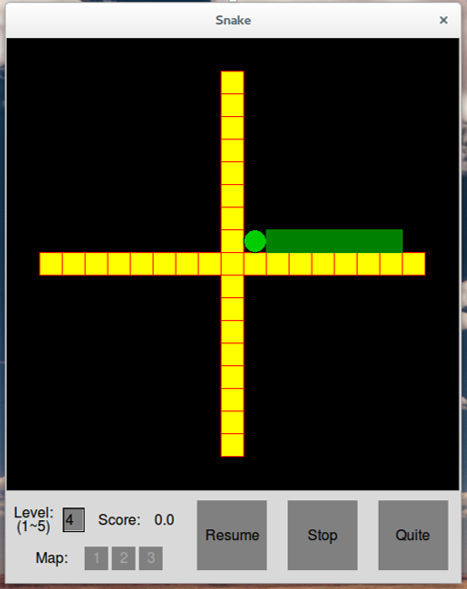
函数main()主要流程如下。首先创建对话框，创建按钮“Quite” “Start” “Pause”，创建三个地图选择按钮，创建等级输入框、成绩显示以及一些静态说明文本。之后进入主while 1循环，每次主循环开始时执行按钮的使能和禁用等准备工作，之后进入游戏前的游戏设定while 1循环。在该循环中可以进行游戏的地图、等级设定，通过getMouse实现。点击对应的地图编号，可以清除之前的地图，构造并绘制新的Wall类，如图六所示。准备状态while 1循环的跳出出口是Quite和Start，前者导致对话框关闭，点击后者之后，进行游戏前的最后准备。进行对一些按钮使能禁用、更改按钮标题的准备性工作，之后构造并绘制Food、Snake实例。完成了准备工作后，就可以进入游戏while 1循环。游戏while 1循环的break情况包括游戏失败、点击“Stop”按钮、点击“Quite”按钮，前种情况下程序重新从主循环开头开始进行，后一种情况下程序又会退出主循环，导致对话框关闭。

游戏循环每进行一次，蛇前进一步，流程如下。使用time.sleep(-0.2\*Level+1.2)延时一段时间，具体时间长度由级别决定。然后使用checkMouse()，如果上次循环之后点击过鼠标，则判断鼠标是否点击到各个按键，其中点击到Pause按钮会改变一个变量pause的值在0和1之间切换，并更改按钮标题，在“Pause”和“Resume”之间切换。如果点击的是游戏界面，且pause为0，游戏正常进行，则调用Click2Turn(clk\_x,clk\_y,head,snk\_x,snk\_y)函数更改蛇的走向。之后再判断如果pause为0，游戏正常进行，则对吃食进行检查后进行移动，之后检查自吃、撞墙，如果出现这两种情况，则调用GameOver函数，并退出游戏循环，否则游戏循环从头再次进行。如果pause为1，则不判断转向、吃食，不检查自吃、撞墙，蛇不会向前移动，如图七所示。

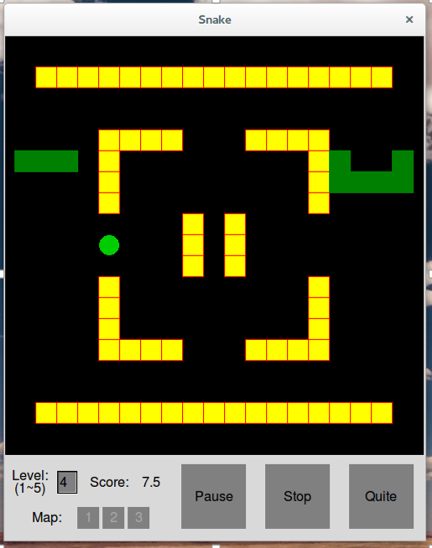
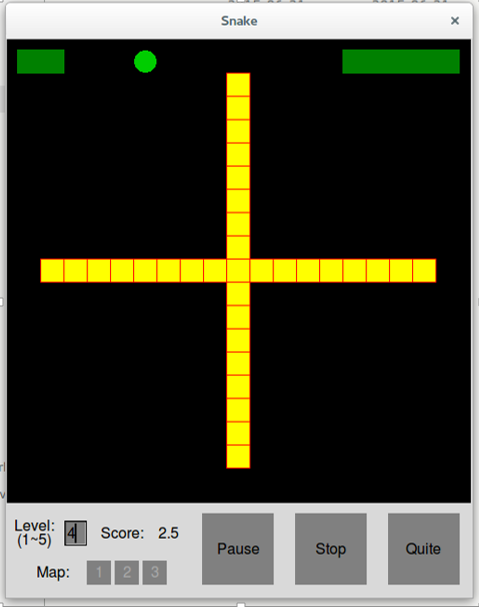
游戏失败调用的GameOver(snake,food,Bt,w)函数先让蛇闪烁两次，之后清除蛇和食物并修改一些按钮标题。



图六、游戏设定while 1循环



图七、游戏暂停状态



图八、游戏进行状态，如果没有墙的阻隔，玩家可以穿越边界