# 综合项目-贪吃蛇

# 程序分析

相信大家都知道贪吃蛇这款游戏，它是一款经典的手机游戏。通过控制蛇头方向吃食物，使得蛇变长，从而获得积分，既简单又耐玩。通过上下左右键控制蛇的方向，寻找吃的东西，每吃一口就能得到一定的积分，而且蛇的身子会越吃越长，身子越长玩的难度就越大，不能碰墙，不能咬到自己的身体，更不能咬自己的尾巴，等到了一定的分数，就能过关，然后继续玩下一关。

## 1、1 模块设计

在本游戏中，从键盘输入来获取移动方向和蛇的移动的关系



## 1、2 模块描述

**蛇的初始化**

蛇的初始化实际就是二维数组的初始化，该二维数组存储两个值，里面包含该蛇身体的坐标信息，它出现的初始位置是横纵坐标的中间位置。

**蛇的移动**

蛇的移动是通改变二维数组坐标位置来实现的，例如当蛇向右前进一个单位，则将尾每一个身体位置的坐标更改，同时改变蛇头、蛇身以及蛇尾的方向。这样整体来看来蛇就前进了一个单位。

**蛇的增长**

当蛇吃了正常食物后，蛇的长度会增加，增加蛇的长度就是在食物的位置增加一个二维数组并且将这个位置变为蛇头。

**蛇的死亡**

当蛇撞上障碍物、自身或者通关时，蛇会死亡，蛇死亡就是二维数组的销毁。

**食物的产生**

食物出现的位置产生都是随机的，这些因素由通过随机函数获取的随机数决定。食物的位置不能出现在障碍物和边界上。

**控制键盘输入**

通过获取键盘输入的W/w(上)、 S/s(下)、 A/a(左)、 D/d(右)来改变蛇模块中移动方向，从而影响蛇的移动方向。

## 1、3 项目分析

游戏界面显示之后，会根据读取的移动方向来控制蛇的移动，蛇移动之后会吃食物，食物被吃掉之后又会随机生成，然后刷新界面，将界面显示出来。

刷新界面时会判断蛇的状态，如果蛇死亡，则结束游戏；如果蛇胜利晋级则进入下一关，重新加载地图，显示新的一关的游戏初始界面；如果蛇没有死亡也没有晋级过关，继续本关游戏，则按照读取的键盘输入控制蛇的移动。

# 2、项目实现

## 2、1 项目创建

**蛇结构体**

用来存放蛇的长度和位置坐标

**蛇移动函数**

用来实现游戏的控制和判断

**显示游戏界面函数**

用来实现游戏的显示

**主函数**

用来实现游戏流程控制

## 2、2 项目设计

**1、蛇结构体**

结构体存储蛇相关信息，代码如下所示：

type Postion struct {

X int

Y int

}

type Snake struct {

pos [WIDE \* HIGH]Postion //位置坐标

dir byte //方向

size int //大小

}

**2、游戏运行**

用来控制游戏相关信息，代码如下所示：

for {

// 程序更新周期，400毫秒

//time.Sleep(time.Microsecond \* 3000)

time.Sleep(time.Second / 3)

//暂停

if s.dir == 'P' {

continue

}

//蛇和边界判断

if s.pos[0].X < 0 || s.pos[0].X >= WIDE || s.pos[0].Y < 0 || s.pos[0].Y >= HIGH {

//将光标移至游戏界面下方

Clib.GotoPostion(0, 23)

return

}

//蛇和身体判断

for i := 1; i < s.size; i++ {

if s.pos[0].X == s.pos[i].X && s.pos[0].Y == s.pos[i].Y {

//将光标移至游戏界面下方

Clib.GotoPostion(0, 23)

return

}

}

//蛇和食物判断

if s.pos[0].X == food.X && s.pos[0].Y == food.Y {

food = Food{Postion{rand.Intn(WIDE), rand.Intn(HIGH)}}

DrawUI(food.Postion, '#')

score++

s.size++

}

//控制蛇的方向

switch s.dir {

case 'U':

dx = 0

dy = -1

case 'L':

dx = -1

dy = 0

case 'R':

dx = 1

dy = 0

case 'D':

dx = 0

dy = 1

}

//记录尾巴位置

lx, ly := s.pos[s.size-1].X, s.pos[s.size-1].Y

//顺序更新每一节蛇的坐标

for i := s.size - 1; i > 0; i-- {

s.pos[i] = s.pos[i-1]

DrawUI(s.pos[i], '\*')

}

//更新蛇头坐标

s.pos[0].X += dx

s.pos[0].Y += dy

DrawUI(s.pos[0], '@')

//去掉尾巴

DrawUI(Postion{lx, ly}, ' ')

}

**3、显示游戏**

用来显示游戏相关信息，代码如下所示：

// 输出初始画面

fmt.Fprintln(os.Stderr,

`

#-----------------------------------------#

| |

| |

| |

| |

| \* |

| |

| |

| |

| |

| |

| |

| |

| |

| |

| |

| |

| |

| |

| |

| |

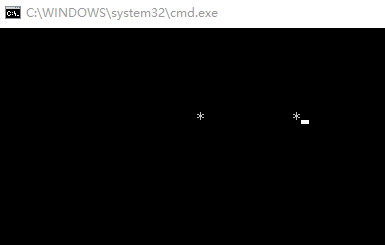
#-----------------------------------------#

`)

## 2、3 项目实现

独立完成代码实现

## 2、4 效果展示



# 3、拓展训练

**3、1 游戏边界**

**3、2 游戏关卡**

**3、3 函数**

kbhit()（VC++6.0下为\_kbhit()）  
　　功能及返回值： 检查当前是否有键盘输入，若有则返回一个非0值，否则返回0  
　　用法：int kbhit(void);  
　　包含头文件： include <conio.h>

getch()　　（VC++6.0下为\_getch()）

功能: 从控制台无回显地取一个字符  
　　用法: int \_getch(void);  
　　返回值：读取的字符  
　　包含头文件： include <conio.h>

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

COORD是Windows API中定义的一种结构，表示一个字符在控制台屏幕上的坐标。

# 4、项目心得

项目内容介绍完毕，将对项目进行简单总结。初学者也应养成这样的习惯，在项目完成后，及时回顾遇到的问题及解决方法，总结得失，为今后的开发工作积累经验。

1. **项目整体规划**

每一个项目，在实现之前都要进行分析设计，项目整体要实现哪些功能。将这些功能划分成不同的模块，如果模块较大还可以在内部划分成更小的功能模块。这样逐个实现每个模块，条理清晰。在实现各个模块后，需要将模块整合，使各个功能协调有序的进行。在进行模块划分和模块整合时，可以使用流程图来表示模块之间的联系与运行流程。

**2、数组的使用**

**3、函数的使用**

**4、面向对象使用**