# 贪吃蛇介绍：

## 功能

游戏地图的绘制

贪吃蛇和食物的打印

通过方向键控制贪吃蛇行进，吃掉食物后，蛇身增长一段

蛇撞墙或者撞到自身死亡，游戏退出

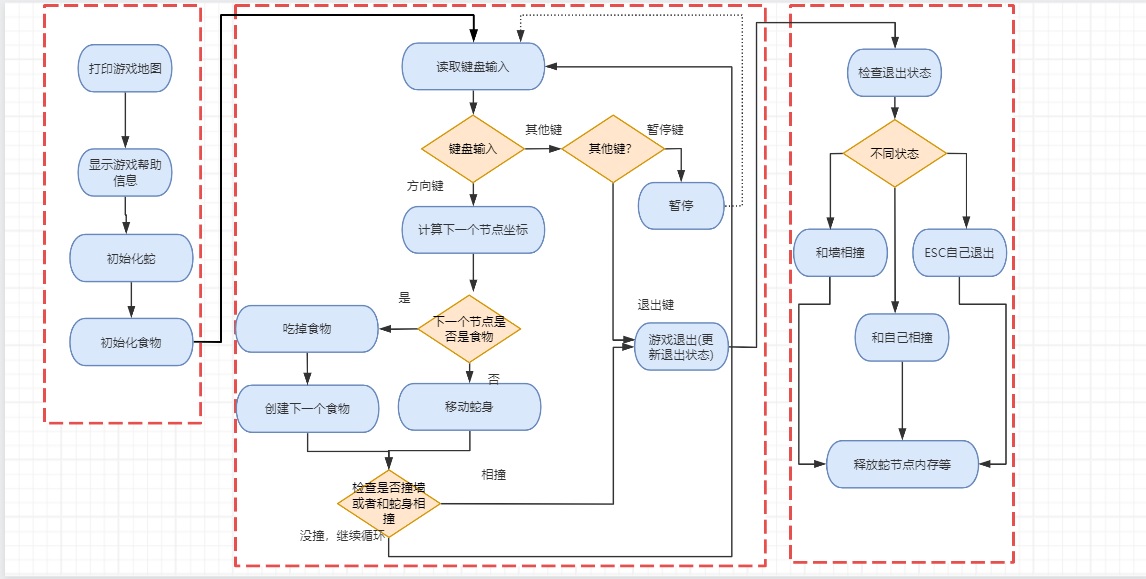
ESC键可以随时退出游戏

空格键SPACE可以随时暂停游戏

F3/F4键可以加快/减慢蛇爬行速度，增加/降低游戏难度

显示游戏当前得分状况

## 流程图



按照流程图所示，游戏可以大致分成三个大模块：

1. 游戏初始化阶段
2. 游戏进行阶段
3. 游戏结束阶段

### 游戏初始化阶段

1. 初始化游戏地图

在初始化阶段，需要在控制台程序上打印游戏的围墙。

尽管Window 提供了API函数，可以在控制台应用程序窗口中打印字符，但API在界面美化上还是有局限，因此这里主要使用EasyX。EasyX 是基于 C++ 的图形界库，可以帮助 C/C++ 初学者快速上手图形和游戏编程。EasyX的使用介绍参考后面章节。

1. 显示游戏的帮助信息

在游戏主界面能显示帮助信息，包括：

* 使用方向键控制蛇移动
* 蛇身体碰墙或者碰到自己，游戏结束
* ESC键退出游戏
* 空格键暂停游戏
* F3加快蛇移动速度
* F4减慢蛇移动速度

1. 初始化蛇和食物

### 游戏进行阶段：

游戏开始后，根据玩家的键盘输入，做不同处理：

1. 方向键（上下左右）

* 获取下一个节点的坐标信息
* 如果下一个节点是食物，吃掉食物并更新玩家得分，然后创建新的食物
* 如果下一个节点不是食物，仅仅移动蛇身。

1. ESC键

* 游戏状态为ESC
* 游戏退出

1. 空格键

* 暂停游戏，用户任意键入后重新开始游戏

1. F3、F4键

* F3加快蛇移动速度
* F4减慢蛇移动速度

1. 判断蛇当前是否撞墙或者撞到自己

* 如果撞墙，更新游戏状态为：KILL\_BY\_WALL
* 如果撞到自己，更新游戏状态为：KILL\_BY\_SELF

1. 显示游戏当前得分情况

### 游戏结束阶段

1. 当游戏的状态不在正常状态（NORMAL）时，游戏退出，包括：

* ESC: 用户自己退出
* KILL\_BY\_WALL：撞墙
* KILL\_BY\_SELF：撞自己

1. 游戏结束前，需要进行的扫尾工作：

* 释放内存，防止内存泄漏

# 二、EasyX使用

## 参考：

<https://www.dotcpp.com/course/easyx-go/>

## 下载和安装

<https://easyx.cn>

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

点击“安装”

安装的本质，是将easyX的.h头文件和.lib库文件复制到Visual C++的include和lib文件夹中。

## 第一个例子: 绘制圆

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

代码如下：

#include <iostream>

//EasyX头文件

#include <graphics.h>

#include <conio.h>

int main()

{

initgraph(640, 480); // 绘制窗口，大小640\*480像素

setfillcolor(YELLOW); // 设置填充颜色为黄色

fillcircle(200, 200, 100); // 画圆，圆心（200，200），半径100

// 按任意键继续

getchar();

closegraph();

return 0;

}

使用Easy X，首先需要进入EasyX的头文件：

//EasyX头文件

#include <graphics.h>

#include <conio.h>

运行结果：

A yellow circle on a black background

Description automatically generated

## 坐标

看到easyX绘图的效果后，普及一下EasyX会用到的几个概念：

坐标

  不仅仅画圆，以及画点、线、圆等各种图案都离不开坐标的概念，那么这个坐标是怎么计算的，那就是以窗口程序的左上角为（0，0）点，x轴和y轴分别向右和向下递增，单位为像素点

A screenshot of a computer

Description automatically generated

需要特别注意的是，一个中文字符占x=15,y=20

## 第二个例子：绘制矩形

代码：

#include <iostream>

//EasyX头文件

#include<graphics.h>

#include<conio.h>

int main()

{

// 绘制窗口，大小640\*480像素

initgraph(640, 480);

// 绘制矩形, 左上点坐标（50，50），右下坐标（200，200）

rectangle(50, 50, 200, 200);

// 按任意键继续

getchar();

closegraph();

return 0;

}

运行结果：

A black screen with a square in it

Description automatically generated

## 第三个例子：加载外部图片并显示

easyX通过loadimage加载图片，然后使用putimage输出到窗口指定位置.

涉及到两个函数：

**loadimage**

第一个参数是传入参数，因此需要加地址符，

第二个参数是图片的路径。 如果使用的路径是相对路径，所以用../。绝对路径当然也可以。需要注意的是，因为C语言中字符串转义字符的存在，这里需要写两个/

第三个/第四个参数：load图片的宽和高

**putimage**

在指定位置输出即可

// 加载图片

IMAGE img;

loadimage(&img, \_T("..//..//Images//Wall3.jpg"), 15, 20, true);

//向（10,10）位置开始输出此图片

putimage(0, 0, &img);

完整代码如下：

#include <iostream>

// EasyX头文件

#include <graphics.h>

#include <conio.h>

int main()

{

// 创建绘图窗口，大小为 640x480 像素

initgraph(900, 500);

// 加载图片

IMAGE img;

loadimage(&img, \_T("..//..//Images//Wall3.jpg"), 15, 20, true);

//向（10,10）位置开始输出此图片

putimage(0, 0, &img);

//任意键继续

getchar();

closegraph();

return 0;

}

输出效果：

A black screen with a white border

Description automatically generated

上面只是显示一张图片的例子，在编写贪吃蛇游戏的时候，会绘制一圈墙，下面就是程序代码：（注意一个中文字符在EasyX坐标中占x=15,y=20）

#include <iostream>

// EasyX头文件

#include <graphics.h>

#include <conio.h>

int main()

{

// 创建绘图窗口，大小为 900x500 像素（900是15倍数，500是20倍数）

initgraph(900, 500);

// 加载图片

IMAGE img;

loadimage(&img, \_T("..//..//Images//Wall3.jpg"), 15, 20, true);

//上面的围墙

for (int i = 0; i<= 900; i=i+15) {

putimage(i, 0, &img);

}

//下面

for (int i = 0; i <= 900; i = i + 15) {

putimage(i, 500-20, &img);

}

//左

for (int i = 20; i < 500; i = i + 20) {

putimage(0, i, &img);

}

//右

for (int i = 20; i < 500; i = i + 20) {

putimage(900 - 15, i, &img);

}

//任意键继续

getchar();

closegraph();

return 0;

}

显示效果：

A black rectangular object with brown squares

Description automatically generated

## 第四个例子：向窗口指定位置输出文本

C语言使用printf函数输出，而且还只能顺着光标的位置打印，如果想特定位置输出还需要不停的输出换行或者回车调整光标位置。而easyX可以很方便的实现这些功能，其函数如下：

**void** outtextxy(**int** x,**int** y,**LPCTSTR** str);

x,y是输出文本的开始坐标

str是输出文本

代码例子：

#include <iostream>

//EasyX

#include <graphics.h>

#include<conio.h>

int main()

{

initgraph(640, 480); // 创建绘图窗口，大小为 640x480 像素

outtextxy(100, 100, \_T("www.dotcpp.com"));

//输出中文

outtextxy(120,120, \_T(“你好“));

getchar(); // 按任意键继续

closegraph(); // 关闭绘图窗口

return 0;

}

以上代码的意思，是向窗口（100，100）位处开始输出“www.dotcpp.com”字符串，默认是白色字体，使用是不是非常方便?

运行效果：

A screen shot of a computer

Description automatically generated

## 如何擦除已经绘制的图案

对于有动态效果的程序，我们为了实现动画效果，会非常频繁的对图案有擦除的需求，对于已经绘制的图案如何擦除呢？可以调用cleardevice()函数清空屏幕，函数原型分别为：

|  |  |
| --- | --- |
| 1 | **void** cleardevice(); |

# 三、主要数据结构

## SnakeNode

蛇节点（包括蛇身和蛇头）可以用下面链表存储：（x,y）记录每个蛇节点的坐标。

/\*

蛇节点用链表结构存储，存储坐标

\*/

struct SnakeNode {

int x;

int y;

SnakeNode\* next;

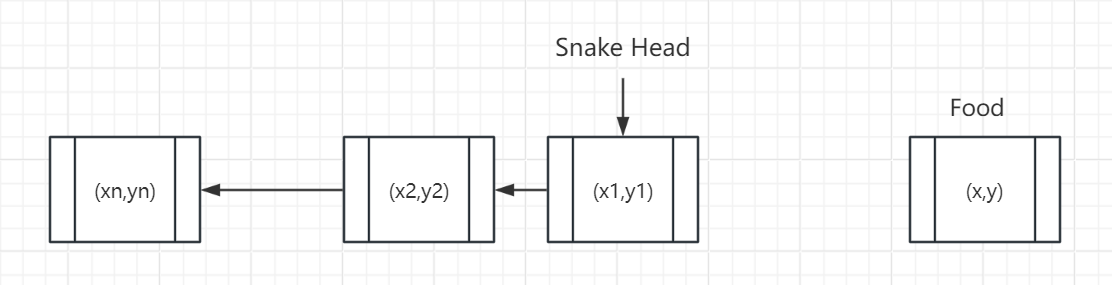
};

这样蛇的吃食物、移动操作转换成链表的插入、删除操作。

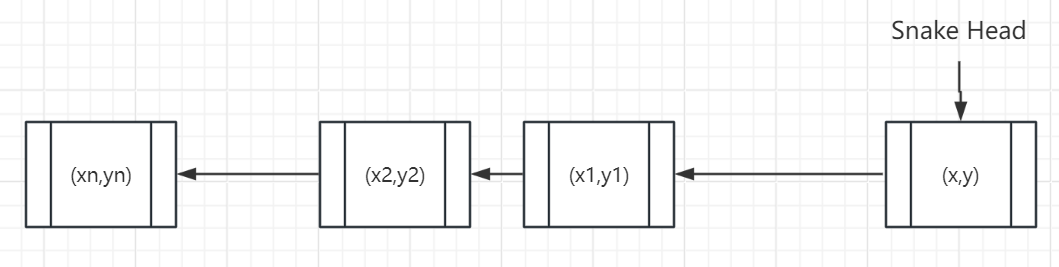
### 如果下一个节点是食物

蛇吃食物就相当与在蛇链表头插入新的节点：

吃食物前：

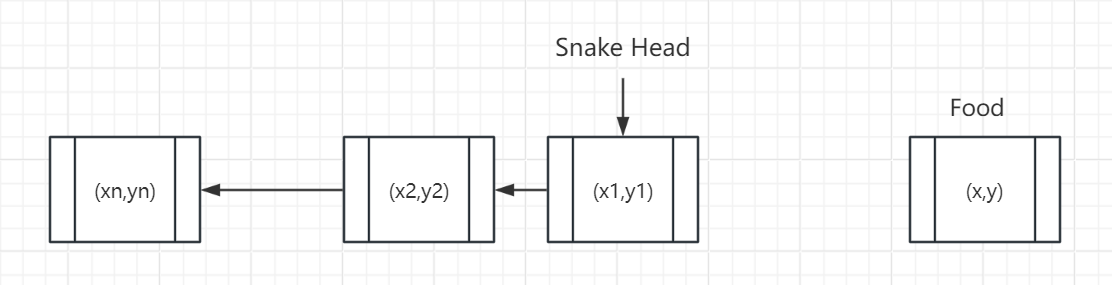


吃掉食物后：



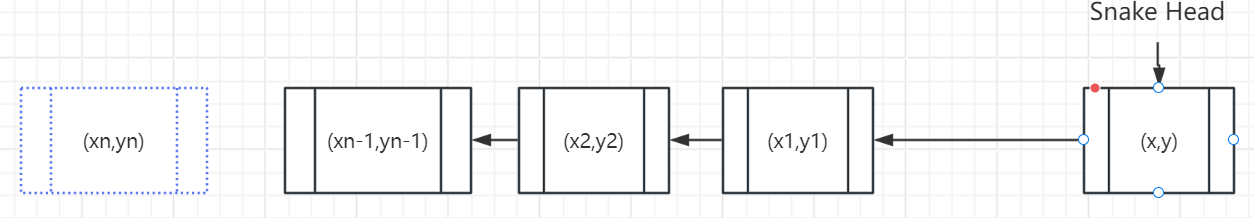
### 如果下一个节点不是食物

蛇移动前：



蛇移动后：

将食物插入蛇头部，蛇尾的的节点从链表中删除（显示的时候需要把image从坐标上擦除）。



## 食物：

当前游戏仅支持一个食物。食物节点主要是也是存储坐标信息，与蛇节点结构类似，因此可以重用SnakeNode，并不需呀重新定义。

struct SnakeNode {

int x;

int y;

SnakeNode\* next;

};

## SnakeGame

SnakeGame是整个游戏的封装，其中包括了蛇、食物、游戏的状态等等。

/\*

游戏封装，包含蛇、食物、状态、score

\*/

struct SnakeGame {

// 蛇节点链表

SnakeNode\* pSnakeNodeList;

// Food

SnakeNode\* pFood;

// 游戏当前状态

GAME\_STATUS status;

// 蛇当前行进的方向

DIRECTION direction;

// 当前Scores

int scores;

// 蛇休眠时间，休眠时间越短，蛇的速度越快，休眠时间越长，蛇的速度越慢，用来控制游戏的难度

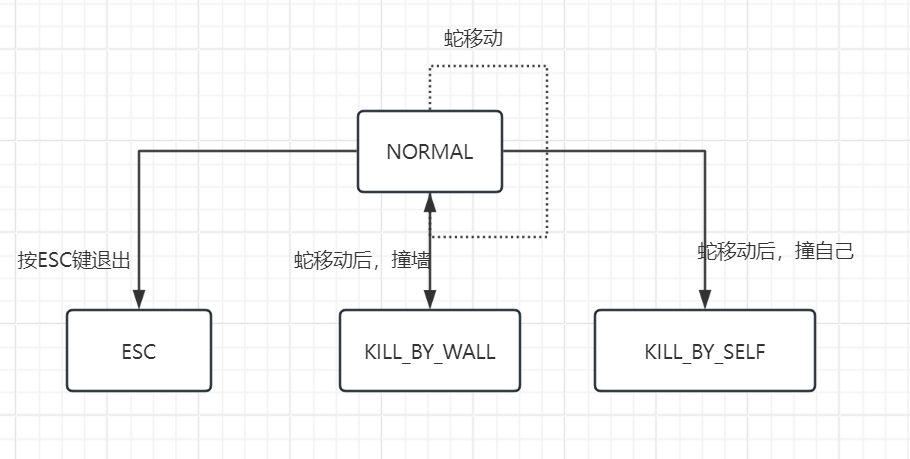
int SleepTime;

};

SnakeGame状态变化如下：

1. 游戏开始后，游戏状态是NORMAL
2. 游戏运行后，状态按照不同输入键变化：

* 如果按ESC键，游戏状态变成ESC，表示玩家主动退出
* 如果是方向键，蛇移动后，蛇头撞到墙，状态变成KILL\_BY\_WALL
* 如果是方向键，蛇移动后，蛇头和蛇身其他节点相交，状态变成KILL\_BY\_SELF

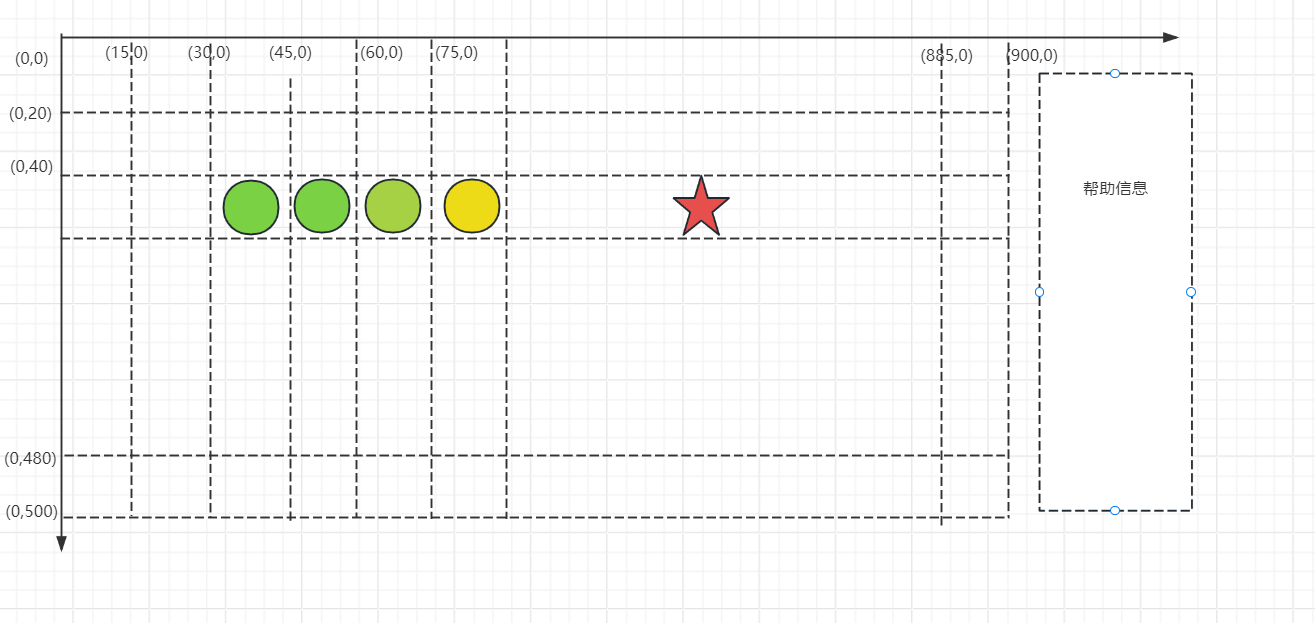


# 四、一些处理

## 绘制蛇和食物

游戏的帮助信息是中文，EasyX支持对汉字的输出，一个汉字占用x=15,y=20像素的块。

假设围墙大小：900\*500(900是15倍数，500是20倍数)，下图是坐标图：



黄色圆：蛇头

绿色圆：蛇身

红色五角：食物

使用loadimage()和putimage()按照不同坐标绘制蛇和食物：

//蛇头

IMAGE img\_snake\_head;

loadimage(&img\_snake\_head, \_T("..//..//Images//snake\_head.jpg"), 15, 20, true);

putimage(30 + 15 + 15 + 15, 40, &img\_snake\_head);

//蛇身

IMAGE img\_snake\_body;

loadimage(&img\_snake\_body, \_T("..//..//Images//snake\_body.jpg"), 15, 20, true);

putimage(30 + 15 + 15, 40, &img\_snake\_body);

putimage(30 + 15, 40, &img\_snake\_body);

putimage(30, 40, &img\_snake\_body);

// 食物

IMAGE img\_food;

loadimage(&img\_food, \_T("..//..//Images//food.jpg"), 15, 20, true);

putimage(150, 40, &img\_food);

完整测试代码：

#include <iostream>

// EasyX头文件

#include <graphics.h>

#include <conio.h>

// 窗口大小（基于节点，15\*75， 20\*25）

#define WINDOW\_WIDTH 75

#define WINDOW\_HEIGHT 25

// 围墙大小(基于节点，15\*60, 20\*25)

#define WALL\_WIDTH 60

#define WALL\_HEIGHT 25

// 围墙大小（基于像素）

#define WALL\_WIDTH\_PIX 15\*WALL\_WIDTH

#define WALL\_HEIGHT\_PIX 20\*WALL\_HEIGHT

int main()

{

// 创建绘图窗口，大小为 15\*75, 20\*25 像素(宽是15倍数，高是20倍数）

initgraph(15\* WINDOW\_WIDTH, 20\*WINDOW\_HEIGHT);

/\*

\*

坐标：

(0,0)---(15,0)---(30,0)---------------------------------(WALL\_WIDTH\_PIX - 15)-

|

|

(0,20)

|

|

(0,40)

|

|

|

|

(0,WALL\_HEIGHT\_PIX-20)

|

撞墙的判断标准：x==0 || x == WALL\_WIDTH\_PIX - 15 || y==0 || y==WALL\_HEIGHT\_PIX-20

\*/

// 加载图片

IMAGE img;

loadimage(&img, \_T("..//..//Images//Wall.jpg"), 15, 20, true);

//上

for (int i = 0; i< WALL\_WIDTH\_PIX; i=i+15) {

putimage(i, 0, &img);

}

//下

for (int i = 0; i < WALL\_WIDTH\_PIX; i = i + 15) {

putimage(i, WALL\_HEIGHT\_PIX -20, &img);

}

//左

for (int i = 20; i < WALL\_HEIGHT\_PIX; i = i + 20) {

putimage(0, i, &img);

}

//右

for (int i = 20; i < WALL\_HEIGHT\_PIX; i = i + 20) {

putimage(WALL\_WIDTH\_PIX - 15, i, &img);

}

// 蛇头

IMAGE img\_snake\_head;

loadimage(&img\_snake\_head, \_T("..//..//Images//snake\_head.jpg"), 15, 20, true);

putimage(30 + 15 + 15 + 15, 40, &img\_snake\_head);

//蛇身

IMAGE img\_snake\_body;

loadimage(&img\_snake\_body, \_T("..//..//Images//snake\_body.jpg"), 15, 20, true);

putimage(30 + 15 + 15, 40, &img\_snake\_body);

putimage(30 + 15, 40, &img\_snake\_body);

putimage(30, 40, &img\_snake\_body);

// 食物

IMAGE img\_food;

loadimage(&img\_food, \_T("..//..//Images//food.jpg"), 15, 20, true);

putimage(150, 40, &img\_food);

//显示帮助信息

outtextxy(WALL\_WIDTH\_PIX+15, 40, \_T("方向键控制蛇移动方向"));

outtextxy(WALL\_WIDTH\_PIX + 15, 60, \_T("F3/F4 : 加速、减速"));

outtextxy(WALL\_WIDTH\_PIX + 15, 80, \_T("ESC键 : 退出"));

outtextxy(WALL\_WIDTH\_PIX + 15, 100, \_T("空格键 : 暂停游戏"));

// score

outtextxy(WALL\_WIDTH\_PIX + 15, 140, \_T("分数 : "));

TCHAR s[4];

\_stprintf\_s(s, \_T("%d"), 99); // 高版本 VC 推荐使用 \_stprintf\_s 函数

outtextxy(WALL\_WIDTH\_PIX + 15, 160, s);

//任意键继续

getchar();

closegraph();

return 0;

}

程序的运行结果：



在这个图中，撞墙的判断标准是：

x==0 || x == WALL\_WIDTH\_PIX - 15 || y==0 || y==WALL\_HEIGHT\_PIX-20

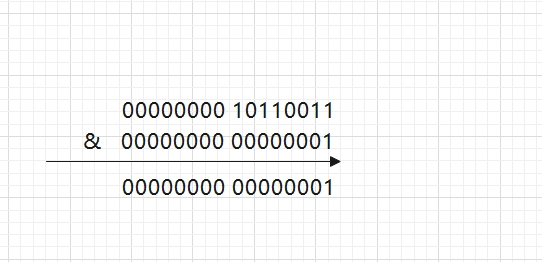
## 如何获取用户按键值

游戏主要是根据玩家的键入值（方向键、空格键等等），来操控蛇和整个游戏进度。那么如何获取用户按键值呢？ Windows提供GetAsyncKeyState函数，这个函数的输入值是键盘上每个键的**虚拟键值，**函数通过返回值来分辨按键的状态。

*这里列出了所有虚拟键值：*[*https://learn.microsoft.com/zh-cn/windows/win32/inputdev/virtual-key-codes*](https://learn.microsoft.com/zh-cn/windows/win32/inputdev/virtual-key-codes)*，例如：VK\_UP、VK\_DOWN、VK\_LEFT、VK\_RIGHT等等。*

GetAsyncKeyState函数返回一个short类型的值，在上⼀次调⽤ GetAsyncKeyState 函数后，如果返回的16位的short数据中，最⾼位是1，说明按键的状态是按下；如果最⾼是0，说明按键的状态是抬起；如果最低位被置为1则说明，该按键被按过，否则为0。 如果我们要判断⼀个键是否被按过，可以检测GetAsyncKeyState返回值的最低值是否为1。

那么如何判断最低为是否为1呢？可以通过与0x01按位与（&）来进行判断



例如下面用来检查5是否被按：

int main()

{

short ret = GetAsyncKeyState(0x50);

if ((ret & 1) == 1)

printf("5 is pressed\n");

else

printf("5 is not pressed\n");

return 0;

}

由于要多次判断不同按键，可以选择用宏定义进行封装，输入虚拟键值，用来检测输入键是否被按下：

//====================

// 虚拟键代码： 可以搜索到常用虚拟键代码

// 如果最低位被置为1则说明，该按键被按过，否则为0。

// 如果我们要判断⼀个键是否被按过，可以检测GetAsyncKeyState返回值的最低值是否为1

//====================

**#define KEY\_PRESS(VK) ((GetAsyncKeyState(VK)&0x1)?1:0)**

测试程序如下：

int main()

{

while (1)

{

if (KEY\_PRESS(VK\_LEFT))

{

printf("LEFT\n");

}

else if (KEY\_PRESS(VK\_RIGHT))

{

printf("RIGHT\n");

}

else if (KEY\_PRESS(VK\_UP))

{

printf("UP\n");

}

else if (KEY\_PRESS(VK\_DOWN))

{

printf("DOWN\n");

}

else if (KEY\_PRESS(VK\_ESCAPE))

{

printf("ESC\n");

}

else if (KEY\_PRESS(VK\_SPACE))

{

printf("SPACE\n");

}

else if (KEY\_PRESS(VK\_F3))

{

printf("F3\n");

}

else if (KEY\_PRESS(VK\_F4))

{

printf("F4\n");

}

}

return 0;

}

## 随机产生新食物

当食物被吃掉后，游戏会产生一个新的食物（随机坐标）。这个食物的坐标是如何是随机产生的呢? 需要用到：srand() 和rand()函数。

仅仅使用rand()函数，因为每次运行的随机数种子是一样的，因此每次运行产生的随机数是相同的。所以需要配合使用srand() ，设置 rand() 产生随机数时的随机数种子，产生真正的随机坐标。系统时间总是在变化的，通常可以用系统时间可以用来设置随机数种子。

示例代码：

#include <iostream>

using namespace std;

void RealRand() {

// 因为time是变化的，用来设置随机数种子

srand((unsigned int)time(NULL));

int x = 0;

//随机数范围：[0,10) : 包括0，不包括10

for (int i = 0; i < 30; i++) {

x = rand() % (10); //随机数范围 (0..9)

cout << "x=" << x << endl;

}

cout << "----------------" << endl;

//[0,10] : 包括0，包括10

for (int i = 0; i < 30; i++) {

x = rand() % (11); //随机数范围 (0..10)

cout << "x=" << x << endl;

}

cout << "----------------" << endl;

//(0,10] : 不包括0，包括10

for (int i = 0; i < 30; i++) {

x = rand() % (10) + 1; //随机数范围 (1..10)

cout << "x=" << x << endl;

}

cout << "----------------" << endl;

//(0,10) : 不包括0，不包括10

for (int i = 0; i < 30; i++) {

x = rand() % (9) + 1; //随机数范围 (1..9)

cout << "x=" << x << endl;

}

cout << "----------------" << endl;

return;

}

int main()

{

// 使用srand设置随机种子，达到真正的随机

RealRand();

return 0;

}

按照范围取随机数，一般有一下几种情况，使用的算式是：

1. 要取得[a, b) 的随机整数（包括a, 不包括b），使用(rand() % (b - a)) + a;

2. 要取得[a, b] 的随机整数（包括a, 也包括b），，使用(rand() % (b - a + 1)) + a;

3. 要取得(a, b] 的随机整数（不包括a, 包括b），，使用(rand() % (b - a)) + a + 1;

4. 要取得(a, b) 的随机整数（不包括a, 也不包括b），，使用(rand() % (b - a - 1)) + a + 1;