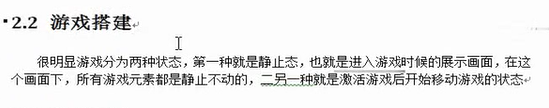
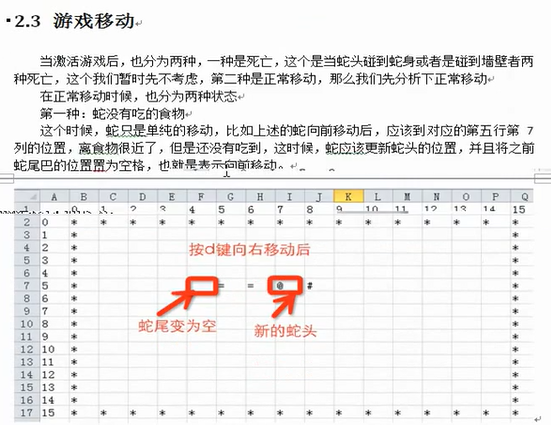
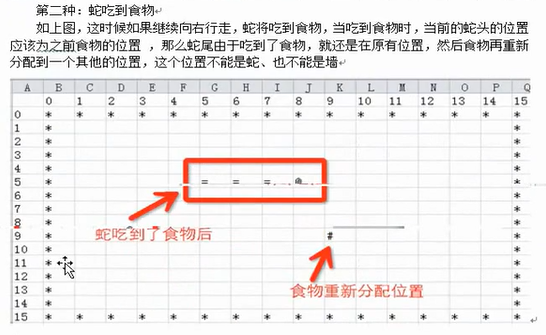
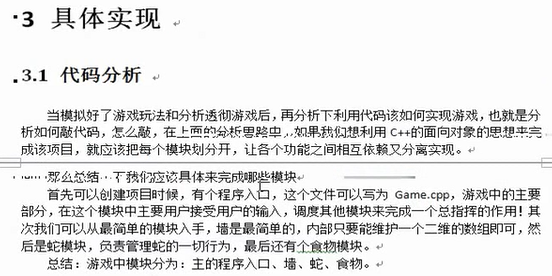


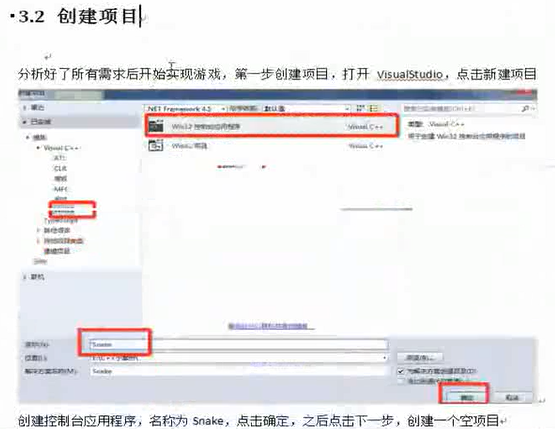
**蛇身一开始可以设置为0，就任何方向都可以激活吧？如果这样蛇头初始位置向上，它的对立边就不能激活吧？**



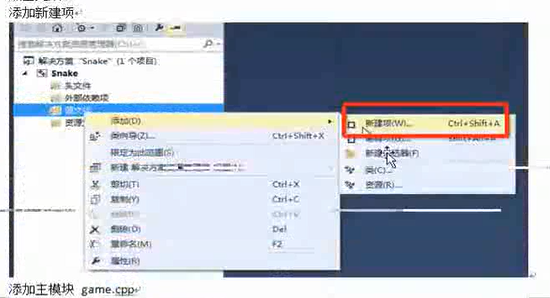


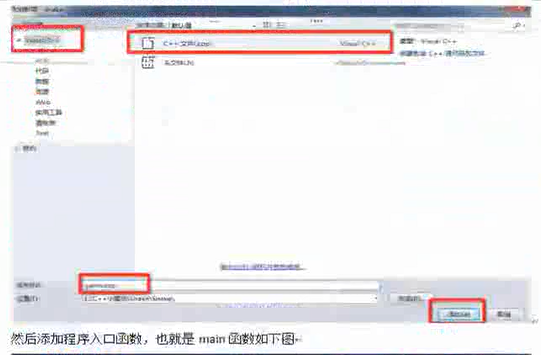


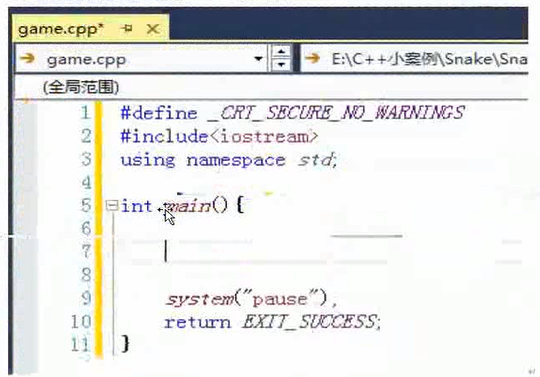


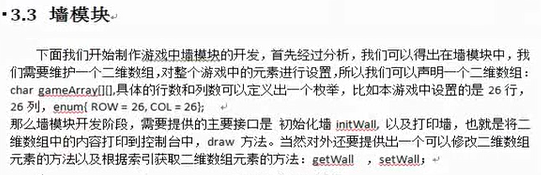


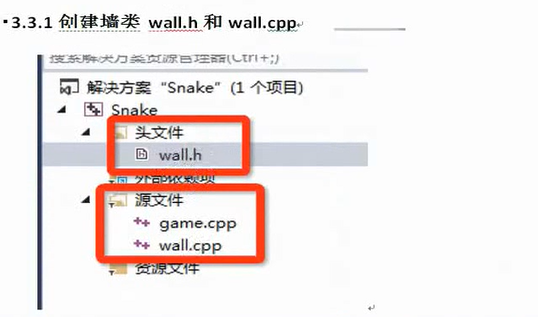


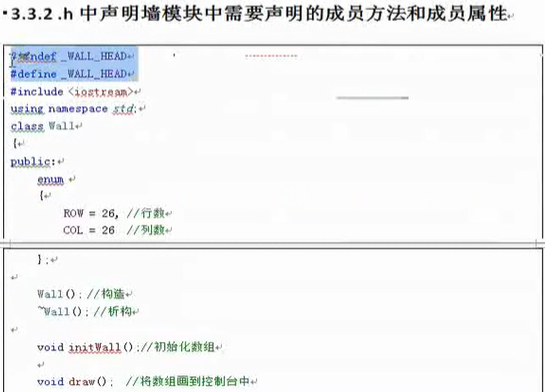














#ifndef \_WALL\_HEAD

#define \_WALL\_HEAD

#include <iostream>

using namespace std;

class Wall {

public:

enum {

ROW = 26,//行

COL = 26//列

};

//初始化墙壁

void initWall();

//展示墙壁

void draw();

//根据索引设置 二维数组里的内容，方便设置蛇的位置

void setWall(int x, int y, char c);

//根据索引获取 当前位置的符号，方便设置蛇的位置

char getWall(int x, int y);

private:

char gameArray[ROW][COL];

};

#endif // !\_WALL\_HEAD

#include "wall.h"

void Wall::initWall()

{

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

for (int j = 0; j < COL; ++j) {

//放墙壁

if (i == 0 || j == 0 || i == ROW - 1 || j == COL - 1) {

gameArray[i][j] = '\*';

}

else

gameArray[i][j] = ' ';

}

}

}

void Wall::draw()

{

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

for (int j = 0; j < COL; ++j) {

cout << gameArray[i][j] << " ";//这里多输出一个空格，墙的显示效果更好

}

//第5行开始输出版本信息

if (i == 5)

cout << "created by CYB";

else if (i == 6)

cout << "A: left";

else if (i == 7)

cout << "D: right";

else if (i == 8)

cout << "W: up";

else if (i == 9)

cout << "S: down";

cout << endl;

}

}

void Wall::setWall(int x, int y, char c)

{

gameArray[x][y] = c;

}

char Wall::getWall(int x, int y)

{

return gameArray[x][y];

}

测试

#include <iostream>

using namespace std;

#include "wall.h"

int main() {

//1、测试墙类中的功能

Wall wall;

wall.initWall();

wall.setWall(5, 5, '=');

wall.setWall(5, 6, '=');

wall.setWall(5, 7, '@');

wall.draw();

cout << wall.getWall(0, 0) << endl;

cout << wall.getWall(5, 5) << endl;

cout << wall.getWall(5, 7) << endl;

cout << wall.getWall(1, 1) << endl;

return 0;

}

---------------------------------------------------------------------------------

#include "wall.h"

class Snake {

public:

Snake(Wall& tempWall);

//节点

struct Point {

//数据域

int x, y;

//指针域

Point\* next;

};

Point\* pHead;//蛇的头结点

Wall& wall;//引用来维护wall，用于初始化设置蛇

//初始化

void initSnake();

//销毁节点

void destroyPoint();

//添加节点

void addPoint(int x, int y);

};

#include "snake.h"

//保证拿到main函数中的同一个wall

Snake::Snake(Wall& tempWall) :wall(tempWall)

{//通过Wall类的对象wall传给tempWall,tempWall初始化Snake类中的wall

//而Snake类中的wall是Wall类的对象wall的引用，所有Snake类的wall就是Wall类的对象wall

pHead = NULL;

}

//销毁所有的节点

void Snake::destroyPoint()

{

Point\* pCur = pHead;

while (pCur != NULL) {//这里用pHead!= NULL同样可以

pCur = pHead->next;

delete pHead;

pHead = pCur;

}

}

void Snake::addPoint(int x, int y)

{

//创建新节点

Point\* newPoint = new Point;

newPoint->x = x;

newPoint->y = y;

newPoint->next = NULL;

//====@ ====@ @ -> = =====@

// 空 = -> @

//如果原来的头不为空，则改为身子=，新的头结点改为头@

if (pHead != NULL) {

wall.setWall(pHead->x, pHead->y, '=');//这里需要使用Wall类中的内容

}

newPoint->next = pHead;

pHead = newPoint;//更新头部

wall.setWall(pHead->x, pHead->y, '@');

// @X X@ =@

//头插法，这里没有传蛇这个链表头参数，而是使用一个全局的pHead头指针，

/\*if (pHead == NULL)

{

pHead = newPoint;

wall.setWall(pHead->x, pHead->y, '@');

}

else

{

pHead->next = newPoint;

Point\* temp = pHead;//记录原头指针

pHead = newPoint;

wall.setWall(pHead->x, pHead->y, '@');

wall.setWall(temp->x, temp->y, '=');

}\*/

}

void Snake::initSnake()

{

destroyPoint();

addPoint(5, 3);

addPoint(5, 4);

addPoint(5, 5);

}

测试

#include "snake.h"

Wall wall;

wall.initWall();

Snake snake(wall);

snake.initSnake();

wall.draw();

---------------------------------------------------------------------------------

#include "wall.h"

class Food {

public:

Food(Wall& temp);

int foodX, foodY;

Wall& wall;//引用来维护wall，用于设置食物

//设置食物

void setFood();

};

#include "food.h"

//保证拿到main函数中的同一个wall

Food::Food(Wall& tempWall) :wall(tempWall)

{

}

void Food::setFood()

{

while (true) {

//保证随机生成的点处于墙内

foodX = rand() % (Wall::ROW - 2) + 1;

foodY = rand() % (Wall::COL - 2) + 1;

//如果随机的位置是蛇头或蛇身 就重新生成随机数，判断只要是空位置就设置食物

if (wall.getWall(foodX, foodY) == ' ') {//这里需要使用Wall类中的内容，使用Wall&wall维护墙中的wall

wall.setWall(foodX, foodY, '#');

break;

}

}

}

测试

#include "food.h"

#include <ctime>

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

Wall wall;

wall.initWall();

Food food(wall);

food.setFood();

wall.draw();

---------------------------------------------------------------------------------

snake.h

#include "food.h"

//删除节点

void delPoint();

enum {

UP = 'w', DOWN = 's', LEFT = 'a', RIGHT = 'd'

};

Food& food;//引用来维护food，用于设置食物

//移动蛇，返回值代表移动是否成功，参数key代表用于移动的方向

bool move(char key);

Snake(Wall& tempWall, Food& tempFood);

snake.cpp

void Snake::delPoint()

{

//尾删：

//两个节点以上才删除

if (pHead == NULL || pHead->next == NULL) {

return;

}

Point\* pCur = pHead->next;

Point\* pPre = pHead;

while (pCur->next != NULL) {

pCur = pCur->next;

pPre = pPre->next;

}

//此时已经移动到尾节点，删除

wall.setWall(pCur->x, pCur->y, ' ');

delete pCur;

pCur = NULL;

pPre->next = NULL;

// prev

// cur

// = = = =

// prev cur

// = = = =

// prev cur

// = = = =

/\*Point\* pCur = pHead;

Point\* pPre = pCur;

while (pCur->next != NULL)

{

pCur = pCur->next;

pPre = pCur;

}

wall.setWall(pCur->x, pCur->y, ' ');

delete pCur;

pPre = NULL;\*/

}

测试

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

Wall wall;

wall.initWall();

Snake snake(wall);

snake.initSnake();

snake.delPoint();

wall.draw();

Snake::Snake(Wall& tempWall, Food& tempFood) :wall(tempWall), food(tempFood)

{

pHead = NULL;

}

bool Snake::move(char key)

{

int x = pHead->x;

int y = pHead->y;

switch (key) {

case UP:

x--;

break;

case DOWN:

x++;

break;

case LEFT:

y--;

break;

case RIGHT:

y++;

break;

default:

break;

}

//判断用户到达位置是否成功

if (wall.getWall(x, y) == '\*' || wall.getWall(x, y) == '=') {

cout << "game over" << endl;

return false;

}

//下一位置有效，可以变换

//到达的位置是#，移动成功分两种，吃到食物和没吃到食物

if (wall.getWall(x, y) == '#') {

addPoint(x, y);

//重新设置食物

food.setFood();

}

else {

//未吃到食物

addPoint(x, y);

delPoint();

}

return true;

}

测试

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

Wall wall;

wall.initWall();

Food food(wall);

Snake snake(wall,food);

snake.initSnake();

snake.move('w');

snake.move('w');

snake.move('a');

wall.draw();

蛇的第一步移动

#include <conio.h>

#include <Windows.h>

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

Wall wall;

wall.initWall();

Food food(wall);

Snake snake(wall, food);

snake.initSnake();

wall.draw();

//接收用户输入

char key = \_getch();//头文件：#include <conio.h>

if (snake.move(key) == true)

{

system("cls");

wall.draw();

Sleep(300);

}

加一个死循环，每次移动都要等待输入

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

Wall wall;

wall.initWall();

Food food(wall);

Snake snake(wall, food);

snake.initSnake();

wall.draw();

//导致死亡后，又能继续玩，无法死亡。而且每次都需要按键

while (true)

{

//接收用户输入

char key = \_getch();//头文件：#include <conio.h>

if (snake.move(key) == true)

{

system("cls");

wall.draw();

Sleep(300);

}

}

如果死亡，结束游戏，如果没有死亡就继续按按键移动，同时只需要按一下按键就一直往一个方向移动

//是否死亡标识

bool isDead = false;

while (!isDead)

{

//接收用户输入

char key = \_getch();//头文件：#include <conio.h>

//按一下，才走一下，希望：只要没有键盘输入一直执行

do

{

if (snake.move(key) == true)

{

system("cls");

wall.draw();

Sleep(300);

}

else

{

isDead = true;

break;

}

} while (!\_kbhit());//没有键盘输入返回真，\_kbhit()当没有键盘输入的时候返回0

}

当游戏最一开始就按了左键，游戏直接结束了，应该防止一开始(第一次)按左键就死亡。之后游戏中按左可以改变方向。

//是否死亡标识

bool isDead = false;

//上一次的方向

char preKey = NULL;

while (!isDead)

{

//接收用户输入

char key = \_getch();//头文件：#include <conio.h>

//防止一开始按左 就死亡，但是这样会导致以后按左，都不能激活

/\*if (key == snake.LEFT)

{

continue;

}\*/

//判断如果是第一次按左键，则不能激活游戏

//判断上一次移动方向

if (preKey == NULL && key == snake.LEFT)

{

continue;

}

//按一下，才走一下，希望：只需要按一下按键就一直往一个方向移动

do

{

preKey = key;//更新preKey

if (snake.move(key) == true)

{

system("cls");

wall.draw();

Sleep(300);

}

else

{

isDead = true;

break;

}

} while (!\_kbhit());//没有键盘输入返回真，\_kbhit()当没有键盘输入的时候返回0

}

当按下Q键出现其他操作，应该只允许按上下左右键。同时在游戏的过程中不能180°掉头移动，应该防止按键冲突

while (!isDead)

{

//接收用户输入

char key = \_getch();//头文件：#include <conio.h>

//防止一开始按左 就死亡，但是这样会导致以后按左，都不能向左移动

/\*if (key == snake.LEFT)

{

continue;

}\*/

//判断如果是第一次按左键，则不能激活游戏。之后游戏中按左可以改变方向。

//判断上一次移动方向

if (preKey == NULL && key == snake.LEFT)

{

continue;

}

//按一下，才走一下，希望：只需要按一下按键就一直往一个方向移动

do

{

//当按q键时，会出问题

//只允许上下左右键

if (key == snake.UP || key == snake.DOWN || key == snake.LEFT || key == snake.RIGHT)

{

//判断本次的按键是否与上次冲突，比如移动方向为右，当按A时应该使本次失效，仍为上次按键方向

if ((key == snake.LEFT && preKey == snake.RIGHT) ||

(key == snake.RIGHT && preKey == snake.LEFT) ||

(key == snake.UP && preKey == snake.DOWN) ||

(key == snake.DOWN && preKey == snake.UP))

{

key = preKey;

}

else

{

preKey = key;//(不是冲突按键)，更新preKey，这样使得只有第一次按键为NULL，满足第一次按左键不能激活游戏

}

if (snake.move(key) == true)

{

system("cls");

wall.draw();

Sleep(300);

}

else

{

isDead = true;

break;

}

}

else

{

key = preKey;//当按了其他按键，都为上一次按键方向

}

} while (!\_kbhit());//没有键盘输入返回真，\_kbhit()当没有键盘输入的时候返回0

}

---------------------------------------------------------------------------------

Snake.h

bool isRool;//判断头部是否碰到尾部构成循环的标识

Snake::Snake(Wall& tempWall, Food& tempFood) :wall(tempWall), food(tempFood)

{

pHead = NULL;

isRool = false;

}

Snake.cpp :: move

当蛇头碰到蛇尾，应当避免游戏结束，而是构成循环。但是当碰到身体其他位置则死亡。

bool Snake::move(char key)

{

int x = pHead->x;

int y = pHead->y;

switch (key)

{

case UP:

x--;

break;

case DOWN:

x++;

break;

case LEFT:

y--;

break;

case RIGHT:

y++;

break;

default:

break;

}

//判断如果下一步碰到的是尾巴，不应该死亡

//找到尾巴pPre

Point\* pCur = pHead->next;

Point\* pPre = pHead;

while (pCur->next != NULL) {

pPre = pPre->next;

pCur = pCur->next;

}

//此时头部将要（下一步）移动到尾节点，则进行循环

if (pCur->x == x && pCur->y) {

//碰到尾巴，循环，即吃掉尾部显示头部

isRool = true;

}

else {

//判断用户到达位置是否成功

if (wall.getWall(x, y) == '\*' || wall.getWall(x, y) == '=')

{

cout << "game over" << endl;

return false;

}

}

//下一位置有效，可以变换

//移动成功分两种，吃到食物和没吃到食物(或没吃到食物，但下一步吃到尾巴)

if (wall.getWall(x, y) == '#') {

addPoint(x, y);

//重新设置食物

food.setFood();//这里需要使用Food类中的内容，使用Food&food来设置食物

}

else {

//未吃到食物

addPoint(x, y);//往前走一步

delPoint();//删除尾节点，会将头设置为空

//当构成循环时，显示头节点

if (isRool == true) {

wall.setWall(x, y, '@');

}

}

return true;

}

当撞墙或撞上除尾部外的身体其他部分后，嵌入墙内或身体，才更想死亡

//此时头部将要（下一步）移动到尾节点，则进行循环

if (pCur->x == x && pCur->y) {

//碰到尾巴，循环，即吃掉尾部显示头部，即吃到了尾部

isRool = true;

//继续

}

else {//头部碰到其他的位置

//判断用户到达位置是否成功

if (wall.getWall(x, y) == '\*' || wall.getWall(x, y) == '=')

{

//当撞墙或撞上除身体外的其他部分后，嵌入墙内或身体，才更象死亡

//当撞到墙，多走一步进入墙，再死亡

addPoint(x, y);//多走一步

delPoint();

system("cls");

wall.draw();

cout << "game over" << endl;

return false;

}

}

当吃到自己除尾部外的其他部位，应当显示嵌入身体，显示尾部

if (wall.getWall(x, y) == '\*')

{

//当撞墙或撞上除身体外的其他部分后，嵌入墙内或身体，才更象死亡

//当撞到墙，多走一步进入墙

addPoint(x, y);//多走一步

delPoint();

system("cls");

wall.draw();

cout << "game over" << endl;

return false;

}

else if (wall.getWall(x, y) == '=')

{

addPoint(x, y);//多走一步

//当撞到自己除尾巴外的其他部位，应当显示嵌入身体，显示尾部，即向前多一步但不删除节点

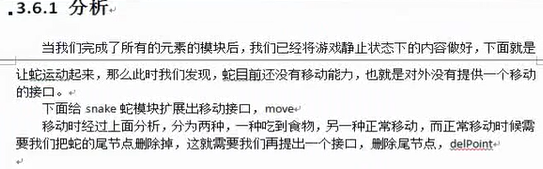
system("cls");

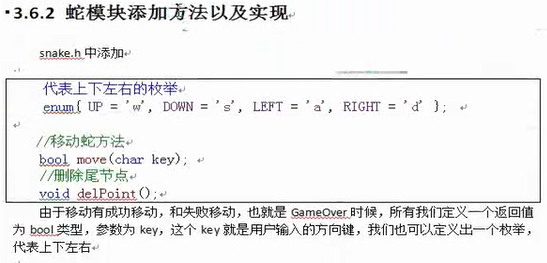
wall.draw();

cout << "game over" << endl;

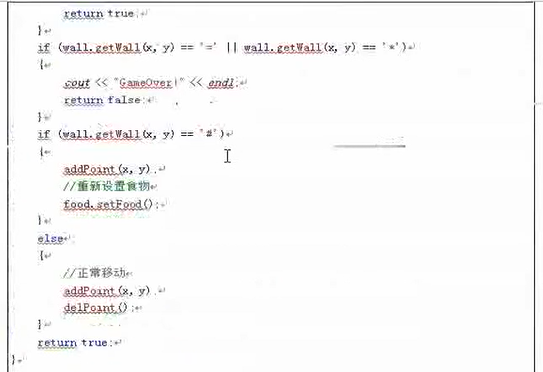
return false;

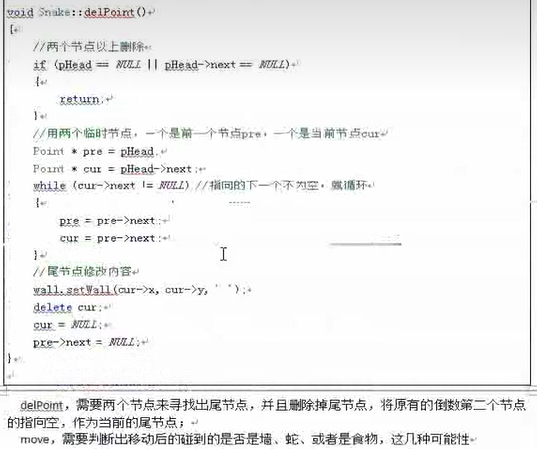
}

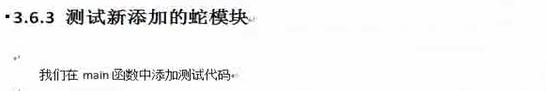




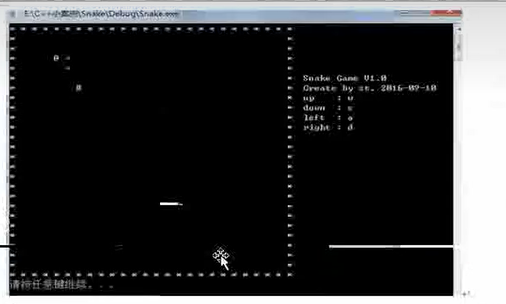


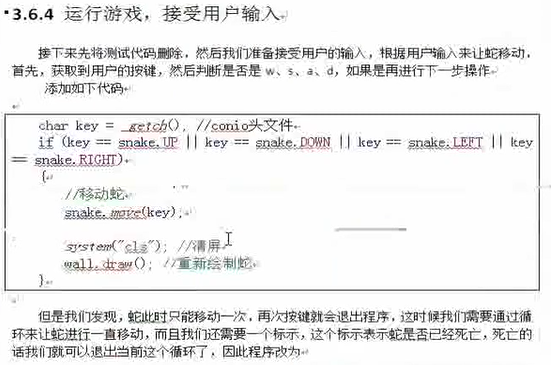








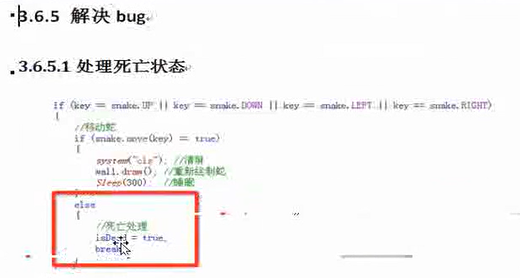


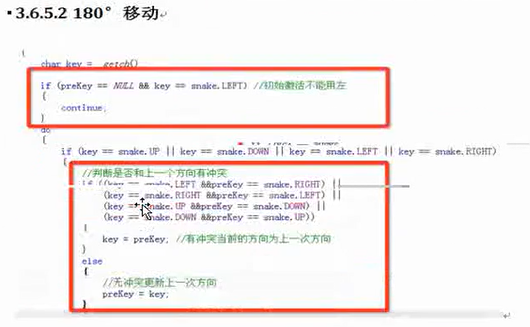


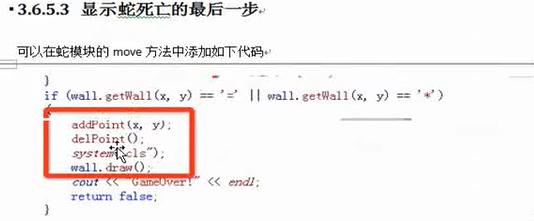


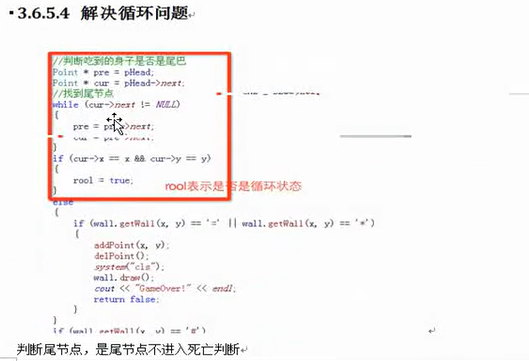


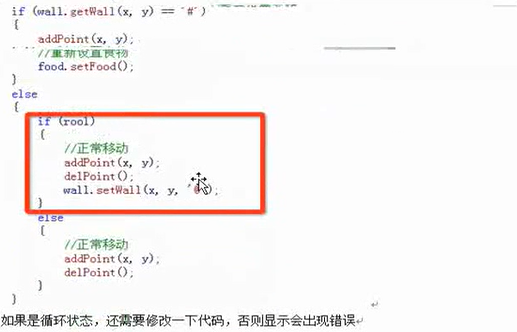


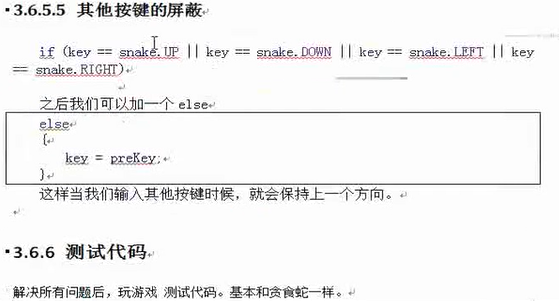












**当蛇撞击左右两侧的墙时不会死亡？但是撞击上下两块墙会死亡**



