TRƯỜNG ĐẠI HỌC XÂY DỰNG

KHOA CÔNG NGHỆ THÔNG TIN

**- - - - 🙡🕮🙣 - - - -**

BÀI TẬP LỚN SỐ 2

**LẬP TRÌNH HỆ THỐNG**

**THE TIMER – GAME RẮN SĂN MỒI**

Sinh viên thực hiện: Lương Văn Sáng

Mã số sinh viên: 1554558

Nhóm môn học: 58PM2

Giáo viên hướng dẫn: Hoàng Nam Thắng

HÀ NỘI 05/2016

**MỤC LỤC**

Bài tập lớn 2 – Lập trình hệ thống

[**I.** **Đặt vấn đề** 2](#_Toc452096017)

[**1.** **Giới thiệu đề tài** 2](#_Toc452096018)

[**2.** **Mục đích** 2](#_Toc452096019)

[**II.** **Giải quyết vấn đề** 2](#_Toc452096020)

[**1.** **Mã nguồn chương trình** 2](#_Toc452096021)

[**2.** **Phân tích chức năng** 12](#_Toc452096022)

[**3.** **Một số hình ảnh** 16](#_Toc452096023)

[**III.** **Kết luận và hướng phát triển** 18](#_Toc452096024)

[**1.** **Kết luận** 18](#_Toc452096025)

[**2.** **Hướng phát triển** 18](#_Toc452096026)

1. **Đặt vấn đề**
2. **Giới thiệu đề tài**

Sau khi nghiên cứu tài liệu AppNee Programming Windows trong chương 8 về phần The Timer, game Rắn săn mồi là bài thực hành đơn giản những kiến thức trong chương cũng như trong sách àm em đã tìm hiểu.

Trong chương trình sử dụng một số kiến thức trong chương khác như bắt sự kiện keyboard.

1. **Mục đích**
   1. Tạo ra được một chương trình game rắn săn mồi có các chức năng cơ bản
   2. Hiểu được cách Timer hoạt động
   3. Hiểu được cách xử lý thông điệp của thủ tục window
2. **Giải quyết vấn đề**
3. **Mã nguồn chương trình**

#include <windows.h>

#include <stdlib.h>

#include <time.h>

#define ID\_TIMER 1

#define REFRESH\_BUTTON 101

#define PAUSE\_BUTTON 102

#define LO\_SPEED\_BUTTON 201

#define MD\_SPEED\_BUTTON 202

#define LG\_SPEED\_BUTTON 203

#define LO\_SPEED 100

#define MD\_SPEED 50

#define LG\_SPEED 30

#define SNAKE\_WIDTH 10

#define MAX\_SIZE 23

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

PSTR szCmdLine, int iCmdShow)

{

static TCHAR szAppName[] = TEXT("Snake");

HWND hwnd;

MSG msg;

WNDCLASS wndclass;

wndclass.style = CS\_HREDRAW | CS\_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

HBRUSH hBrush = CreateSolidBrush(RGB(127, 219, 255));

wndclass.hbrBackground = hBrush;

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

if (!RegisterClass(&wndclass))

{

MessageBox(NULL, TEXT("This program requires Windows NT!"),

szAppName, MB\_ICONERROR);

return 0;

}

hwnd = CreateWindow(szAppName, // window class name

TEXT("Rắn săn mồi"), // window caption

WS\_OVERLAPPEDWINDOW, // window style

CW\_USEDEFAULT, // initial x position

CW\_USEDEFAULT, // initial y position

CW\_USEDEFAULT, // initial x size

CW\_USEDEFAULT, // initial y size

NULL, // parent window handle

NULL, // window menu handle

hInstance, // program instance handle

NULL); // creation parameters

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

DeleteObject(hBrush);

return msg.wParam;

}

void GetRect(HWND hwnd, LPRECT rect)

{

// Lấy thông tin khung giới hạn vùng di chuyển và tính toán sao cho rắn đi được hết hàng và cột.

GetClientRect(hwnd, rect);

rect->left = 130 + SNAKE\_WIDTH;

rect->top = SNAKE\_WIDTH;

rect->right = rect->right - ((rect->right - rect->left) % SNAKE\_WIDTH);

rect->bottom = rect->bottom - (rect->bottom % SNAKE\_WIDTH);

}

int random(int min, int max)

{

int r = rand();

int n = (max - min) / SNAKE\_WIDTH;

int rs = (r % n) \* SNAKE\_WIDTH + min;

return rs;

}

POINT getNewPoint(LPRECT rect, POINT snake[], int size)

{

// Tạo nguồn để tính ra số ngẫu nghiên theo thuật toán của hàm rand()

srand(time(NULL));

POINT point;

// Chọn toạ độ ngẫu nhiên cho mồi

while (TRUE)

{

point.x = random(rect->left, rect->right, SNAKE\_WIDTH);

point.y = random(rect->top, rect->bottom, SNAKE\_WIDTH);

//Kiểm tra xem thức ăn có thuộc vào rắn

int i = 0;

while (i < size)

{

if (point.x == snake[i].x && point.y == snake[i].y)

break;

i++;

}

// Nếu i vượt qua độ dài của rắn thì mồi không thuộc rắn --> thoát vòng lặp

if (i >= size) break;

}

return point;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

PAINTSTRUCT ps;

// Lưu thông tin khung giới hạn vùng di chuyển

static RECT rect;

// Vùng hiện điểm

static RECT score\_rect;

// Các điểm của rắn

static POINT snake[MAX\_SIZE];

// Độ dài của rắn. Khởi tạo = 3

static int size = 3;

// Tốc độ

static int speed = MD\_SPEED;

// Lưu điểm làm mồi

static POINT point;

// Hướng di chuyển

static int direction = VK\_RIGHT;

// Cờ đánh dấu có đang chơi hay không

static BOOL playing = FALSE;

// Điểm số

static int score = 0;

// Hàng đợi hướng di chuyển.

// Có thể bấm phím di chuyển nhanh quá, chương trình chưa kịp xử lý nên phải cho vào hàng đợi

static int queue\_dir[20];

// Kích thước hàng đợi

static int count\_queue = 0;

switch (message)

{

case WM\_CREATE:

// Lấy thông tin khung giới hạn vùng di chuyển

GetRect(hwnd, &rect);

// Tạo các nút

CreateWindow(L"BUTTON", L"CHẠY / DỪNG", WS\_VISIBLE | WS\_CHILD, 25, 20, 100, 24, hwnd, (HMENU)PAUSE\_BUTTON, GetModuleHandle(NULL), NULL);

CreateWindow(L"BUTTON", L"LÀM MỚI", WS\_VISIBLE | WS\_CHILD, 25, 146, 100, 24, hwnd, (HMENU)REFRESH\_BUTTON, GetModuleHandle(NULL), NULL);

CreateWindow(L"BUTTON", L"CHẬM", WS\_VISIBLE | WS\_CHILD, 25, 180, 100, 24, hwnd, (HMENU)LO\_SPEED\_BUTTON, GetModuleHandle(NULL), NULL);

CreateWindow(L"BUTTON", L"VỪA", WS\_VISIBLE | WS\_CHILD, 25, 210, 100, 24, hwnd, (HMENU)MD\_SPEED\_BUTTON, GetModuleHandle(NULL), NULL);

CreateWindow(L"BUTTON", L"NHANH", WS\_VISIBLE | WS\_CHILD, 25, 240, 100, 24, hwnd, (HMENU)LG\_SPEED\_BUTTON, GetModuleHandle(NULL), NULL);

// Khởi tạo con rắn

snake[2].x = rect.left + SNAKE\_WIDTH;

snake[2].y = rect.top + SNAKE\_WIDTH \* 5;

snake[1].x = snake[2].x + SNAKE\_WIDTH;

snake[1].y = snake[2].y;

snake[0].x = snake[1].x + SNAKE\_WIDTH;

snake[0].y = snake[2].y;

// Khởi tạo mồi

point = getNewPoint(&rect, snake, size);

// Khởi tạo vùng hiện điểm

score\_rect.left = 25;

score\_rect.top = 45;

score\_rect.right = 125;

score\_rect.bottom = 145;

return 0;

case WM\_SIZE:

// Lấy thông tin khung giới hạn vùng di chuyển

GetRect(hwnd, &rect);

// Vẽ lại vùng Client

InvalidateRect(hwnd, NULL, TRUE);

return 0;

case WM\_KEYDOWN:

// Bắt phím di chuyển khi đang chơi

if (playing == TRUE)

{

int last\_dir;

if (count\_queue == 0) last\_dir = direction;

else last\_dir = queue\_dir[count\_queue - 1];

// Kiểm tra phím di chuyển hợp lệ mới lưu vào hàng đợi

if ((wParam == VK\_LEFT || wParam == VK\_RIGHT) && (last\_dir == VK\_LEFT || last\_dir == VK\_RIGHT))

return 0;

if ((wParam == VK\_UP || wParam == VK\_DOWN) && (last\_dir == VK\_UP || last\_dir == VK\_DOWN))

return 0;

// Lưu phím di chuyển vào hàng đợi

switch (wParam)

{

case VK\_LEFT:

case VK\_UP:

case VK\_RIGHT:

case VK\_DOWN:

queue\_dir[count\_queue] = wParam;

count\_queue++;

break;

}

}

return 0;

case WM\_COMMAND:

// Bắt sự kiện bấm các nút

if (playing == TRUE && LOWORD(wParam) != PAUSE\_BUTTON)

{

MessageBoxW(hwnd, TEXT("Game chưa kết thúc!"), TEXT("Cảnh báo"), 0);

break;

}

switch (LOWORD(wParam))

{

case PAUSE\_BUTTON:

if (playing == FALSE)

{

// Đánh dấu đang chơi

playing = TRUE;

// Đặt timer gửi thông điệp WM\_TIMER để chạy tiếp game

SetTimer(hwnd, ID\_TIMER, speed, NULL);

}

else

{

// Huỷ timer để dừng game

KillTimer(hwnd, ID\_TIMER);

playing = FALSE;

}

break;

case LO\_SPEED\_BUTTON:

speed = LO\_SPEED;

MessageBoxW(hwnd, TEXT("Tốc độ: Chậm!"), TEXT("Cảnh báo"), 0);

break;

case MD\_SPEED\_BUTTON:

speed = MD\_SPEED;

MessageBoxW(hwnd, TEXT("Tốc độ: Vừa!"), TEXT("Cảnh báo"), 0);

break;

case LG\_SPEED\_BUTTON:

speed = LG\_SPEED;

MessageBoxW(hwnd, TEXT("Tốc độ: Nhanh!"), TEXT("Cảnh báo"), 0);

break;

case REFRESH\_BUTTON:

if (playing == FALSE)

{

// Thiết đặt lại các giá trị ban đầu

size = 3;

score = 0;

playing = FALSE;

direction = VK\_RIGHT;

count\_queue = 0;

// Khởi tạo rắn

snake[2].x = rect.left + SNAKE\_WIDTH;

snake[2].y = rect.top + SNAKE\_WIDTH \* 5;

snake[1].x = snake[2].x + SNAKE\_WIDTH;

snake[1].y = snake[2].y;

snake[0].x = snake[1].x + SNAKE\_WIDTH;

snake[0].y = snake[2].y;

// Khởi tạo mồi

point = getNewPoint(&rect, snake, size);

// Vẽ lại vùng Client

InvalidateRect(hwnd, NULL, TRUE);

}

break;

}

// SetFocus cho cửa sổ chính để nhận được các thông điệp bàn phím

SetFocus(hwnd);

return 0;

case WM\_TIMER:

switch (wParam)

{

case ID\_TIMER:

{

// Vùng cần vẽ lại

RECT rct;

// Cờ kiểm tra xem rắn "chết" hay chưa

BOOL alive = TRUE;

// Kiểm tra xem rắn đâm vào cạnh không

if (snake[0].x >= rect.right || snake[0].x < rect.left || snake[0].y >= rect.bottom || snake[0].y < rect.top)

{

alive = FALSE;

}

// Kiểm tra xem rắn có đâm vào thân hay không

for (int i = 3; i < size; i++)

{

if (snake[0].x == snake[i].x && snake[0].y == snake[i].y)

{

alive = FALSE;

}

}

// Nếu rắn "chết" thì ngừng timer và đưa ra thông báo

if (alive == FALSE)

{

// Bỏ dấu cờ đang chơi

playing = FALSE;

KillTimer(hwnd, ID\_TIMER);

MessageBoxW(hwnd, TEXT("Thua cuộc!"), TEXT("Kết quả"), 0);

return 0;

}

// Kiểm tra rắn có ăn mồi hay không

if (snake[0].x == point.x && snake[0].y == point.y)

{

// Nếu có thì tăng kích thước rắn

size++;

// Lùi các phần tử của rắn lại 1 đơn vị

for (int i = size - 1; i > 0; i--)

{

snake[i] = snake[i - 1];

}

// Gán đầu rắn là mồi vừa ăn

snake[0] = point;

// Tăng điểm

score++;

// Vẽ lại vùng hiển thị điểm

InvalidateRect(hwnd, &score\_rect, TRUE);

// Lấy vùng hiển thị mồi cũ

rct.left = point.x;

rct.top = point.y;

rct.right = point.x + SNAKE\_WIDTH;

rct.bottom = point.y + SNAKE\_WIDTH;

// Vẽ lại vùng của mồi cũ để xoá đi

InvalidateRect(hwnd, &rct, TRUE);

// Tính lại toạ độ con mồi khác

point = getNewPoint(&rect, snake, size);

// Lấy vùng hiển thị mồi mới

rct.left = point.x;

rct.top = point.y;

rct.right = point.x + SNAKE\_WIDTH;

rct.bottom = point.y + SNAKE\_WIDTH;

// Vẽ lại vùng của mồi mới để hiển thị

InvalidateRect(hwnd, &rct, TRUE);

}

// Nếu rắn đạt độ dài tối đa cho phép thì ngừng timer và đưa ra thông báo

if (size == MAX\_SIZE)

{

playing = FALSE;

KillTimer(hwnd, ID\_TIMER);

MessageBox(hwnd, TEXT("Rắn đã đạt độ dài tối đa cho phép. Bạn thắng!"), TEXT("Thông báo"), 0);

return 0;

}

// Dịch chuyển

// Lấy vùng hiển thị đuôi rắn

rct.left = snake[size-1].x;

rct.top = snake[size - 1].y;

rct.right = snake[size - 1].x + SNAKE\_WIDTH;

rct.bottom = snake[size - 1].y + SNAKE\_WIDTH;

// Vẽ lại vùng đuôi rắn để xoá đi

InvalidateRect(hwnd, &rct, TRUE);

// Gán lại toạn độ các phần tử phía sau bằng phần tử đứng trước nó.

for (int i = size - 1; i > 0; i--)

snake[i] = snake[i - 1];

// Lấy hướng đi trong hàng đợi hướng đi, nếu hết rồi thì giữ nguyên hướng trước đó.

if (count\_queue > 0)

{

if ((queue\_dir[0] >= 37 && queue\_dir[0] <= 40) &&

!((direction == VK\_LEFT || direction == VK\_RIGHT) &&

(queue\_dir[0] == VK\_LEFT || queue\_dir[0] == VK\_RIGHT)) &&

!((direction == VK\_UP || direction == VK\_DOWN) &&

(queue\_dir[0] == VK\_UP || queue\_dir[0] == VK\_DOWN)))

{

direction = queue\_dir[0];

}

count\_queue--;

for (int i = 0; i < count\_queue; i++) queue\_dir[i] = queue\_dir + 1;

}

switch (direction)

{

case VK\_LEFT:

snake[0].x -= SNAKE\_WIDTH;

break;

case VK\_UP:

snake[0].y -= SNAKE\_WIDTH;

break;

case VK\_RIGHT:

snake[0].x += SNAKE\_WIDTH;

break;

case VK\_DOWN:

snake[0].y += SNAKE\_WIDTH;

break;

}

// Lưu lại vùng hiển thị của đầu rắn

rct.left = snake[0].x;

rct.top = snake[0].y;

rct.right = snake[0].x + SNAKE\_WIDTH;

rct.bottom = snake[0].y + SNAKE\_WIDTH;

// Vẽ lại vùng đầu rắn để hiển thị ở vị trí mới

InvalidateRect(hwnd, &rct, TRUE);

break;

}

}

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

// Hiển thị điểm

Rectangle(hdc, score\_rect.left, score\_rect.top, score\_rect.right, score\_rect.bottom);

TextOutW(hdc, 60, 50, TEXT("Điểm"), lstrlen(TEXT("Điểm")));

HFONT hFont = CreateFont(50, 0, 0, 0, FW\_SEMIBOLD, 0, 0, 0, 0, 0, 0, 0, 0, L"Consolas");

SelectObject(hdc, hFont);

wchar\_t diem[10];

wsprintf(diem, TEXT("%3d"), score);

TextOutW(hdc, 35, 80, diem, lstrlen(diem));

DeleteObject(hFont);

// Vẽ khung giới hạn trò chơi

Rectangle(hdc, rect.left, rect.top, rect.right, rect.bottom);

// Vẽ rắn

HBRUSH hBrushGreen = CreateSolidBrush(RGB(46, 204, 64));

SelectObject(hdc, hBrushGreen);

for (int i = 0; i < size; i++)

{

Rectangle(hdc, snake[i].x, snake[i].y, snake[i].x + SNAKE\_WIDTH, snake[i].y + SNAKE\_WIDTH);

}

// Vẽ mồi

HBRUSH hBrushRed = CreateSolidBrush(RGB(255, 65, 54));

SelectObject(hdc, hBrushRed);

Rectangle(hdc, point.x, point.y, point.x + SNAKE\_WIDTH, point.y + SNAKE\_WIDTH);

DeleteObject(hBrushRed);

DeleteObject(hBrushGreen);

EndPaint(hwnd, &ps);

return 0;

case WM\_DESTROY:

KillTimer(hwnd, ID\_TIMER);

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}

1. **Phân tích chức năng**
   1. *Hàm lấy 1 số ngẫu nhiên trong 1 đoạn [min,max] cho trước*

int random(int min, int max)

{

int r = rand();

int n = (max - min) / SNAKE\_WIDTH;

int rs = (r % n) \* SNAKE\_WIDTH + min;

return rs;

}

Đầu tiên khai báo 1 biến r, lấy giá trị là 1 số ngẫu nhiên trong khoảng 1 đến 32767.

Sau đó tính n là số vị trí tối đa mà mồi / rắn có thể đặt vào. Sau đó tính vị trí thật của mồi / rắn và trả về cho hàm.

* 1. *Hàm lấy toạ độ ngẫu nhiên cho mồi*

Đầu tiên phải tạo nguồn để tính ra số ngẫu nghiên theo thuật toán của hàm rand() bằng cách dùng hàm srand(time(NULL));

Sau đó, tính toạ độ ngẫu nhiên cho điểm góc trên bên trái của mồi, nếu trùng với thân rắn thì tính lại cho đến khi mồi không thuộc rắn nữa thì thoát khỏi vòng lặp chọn toạ độ.

* 1. *Chức năng hiện điểm*

Khởi tạo 1 vùng để hiển thị điểm:

score\_rect.left = 25;

score\_rect.top = 45;

score\_rect.right = 125;

score\_rect.bottom = 145;

Hiển thị điểm:

Rectangle(hdc, score\_rect.left, score\_rect.top, score\_rect.right, score\_rect.bottom);

TextOutW(hdc, 60, 50, TEXT("Điểm"), lstrlen(TEXT("Điểm")));

HFONT hFont = CreateFont(50, 0, 0, 0, FW\_SEMIBOLD, 0, 0, 0, 0, 0, 0, 0, 0, L"Consolas");

SelectObject(hdc, hFont);

wchar\_t diem[10];

wsprintf(diem, TEXT("%3d"), score);

TextOutW(hdc, 35, 80, diem, lstrlen(diem));

DeleteObject(hFont);

* 1. *Chức năng hiển thị rắn*

Vẽ các ô vuông cấu tạo thành thân rắn. Thông tin của rắn là một mảng các toạ độ điểm trên bên trái của mỗi ô vuông tạo nên rắn.

HBRUSH hBrushGreen = CreateSolidBrush(RGB(46, 204, 64));

SelectObject(hdc, hBrushGreen);

for (int i = 0; i < size; i++)

{

Rectangle(hdc, snake[i].x, snake[i].y, snake[i].x + SNAKE\_WIDTH, snake[i].y + SNAKE\_WIDTH);

}

* 1. *Chức năng chọn tốc độ Chậm, Vừa, Nhanh*

Có 3 nút để thay đổi tốc độ. Bản chất ở đây là thay đổi khoảng thời gian giữa mỗi lần gửi thông điệp WM\_TIMER, thay đổi bằng cách thay đổi speem thành các giá trị khác nhau, để khi gọi lại hàm SetTimer và truyền vào speed thì các khoảng thời gian sẽ khác nhau cho cùng 1 ID Timer.

case LO\_SPEED\_BUTTON:

speed = LO\_SPEED;

MessageBoxW(hwnd, TEXT("Tốc độ: Chậm!"), TEXT("Cảnh báo"), 0);

break;

case MD\_SPEED\_BUTTON:

speed = MD\_SPEED;

MessageBoxW(hwnd, TEXT("Tốc độ: Vừa!"), TEXT("Cảnh báo"), 0);

break;

case LG\_SPEED\_BUTTON:

speed = LG\_SPEED;

MessageBoxW(hwnd, TEXT("Tốc độ: Nhanh!"), TEXT("Cảnh báo"), 0);

break;

* 1. *Chức năng kiểm tra rắn có bị “chết” hay không*

Rắn bị “chết” khi đâm vào tường hoặc đâm vào thân mình.

// Cờ kiểm tra xem rắn "chết" hay chưa

BOOL alive = TRUE;

// Kiểm tra xem rắn đâm vào cạnh không

if (snake[0].x >= rect.right || snake[0].x < rect.left || snake[0].y >= rect.bottom || snake[0].y < rect.top)

{

alive = FALSE;

}

// Kiểm tra xem rắn có đâm vào thân hay không

for (int i = 3; i < size; i++)

{

if (snake[0].x == snake[i].x && snake[0].y == snake[i].y)

{

alive = FALSE;

}

}

* 1. *Chức năng CHẠY / DỪNG*

case PAUSE\_BUTTON:

if (playing == FALSE)

{

// Đánh dấu đang chơi

playing = TRUE;

// Đặt timer gửi thông điệp WM\_TIMER để chạy tiếp game

SetTimer(hwnd, ID\_TIMER, speed, NULL);

}

else

{

// Huỷ timer để dừng game

KillTimer(hwnd, ID\_TIMER);

playing = FALSE;

}

break;

Khi click vào nút CHẠY / DỪNG, chương trình sẽ kiểm tra biến playing (biến đánh dấu xem rắn có đang chạy hay không). Nếu playing == FALSE, tức là rắn đang dừng thì gán lại biến playing = TRUE rồi gọi hàm SetTimer để gửi thông điệp WM\_TIMER để rắn chạy. Ngược lại, playing == TRUE tức là rắn đang chạy thì huỷ timer để dừng rắn bằng cách gọi hàm KillTimer, sau đó gán lại biến playing = FALSE.

* 1. *Xử lý thông điệp WM\_KEYDOWN bắt phím di chuyển*

// Bắt phím di chuyển khi đang chơi

if (playing == TRUE)

{

int last\_dir;

if (count\_queue == 0) last\_dir = direction;

else last\_dir = queue\_dir[count\_queue - 1];

// Kiểm tra phím di chuyển hợp lệ mới lưu vào hàng đợi

if ((wParam == VK\_LEFT || wParam == VK\_RIGHT) && (last\_dir == VK\_LEFT || last\_dir == VK\_RIGHT))

return 0;

if ((wParam == VK\_UP || wParam == VK\_DOWN) && (last\_dir == VK\_UP || last\_dir == VK\_DOWN))

return 0;

// Lưu phím di chuyển vào hàng đợi

switch (wParam)

{

case VK\_LEFT:

case VK\_UP:

case VK\_RIGHT:

case VK\_DOWN:

queue\_dir[count\_queue] = wParam;

count\_queue++;

break;

}

}

return 0;

Chỉ bắt phím di chuyển khi rắn đang chạy (playing == TRUE). Sẽ dùng 1 biến last\_dir để lưu phím cuối cùng vừa bấm trước đó, sau đó kiểm tra phím vừa bấm có hợp lệ không (không được ngược hướng với phím trước đó). Nếu phím vừa bấm là hợp lệ thì sẽ đẩy nó vào cuối hàng đợi hướng đi. Cần có 1 hàng đợi hướng đi vì trong khoảng thời gian giữa 2 thông điệp WM\_TIMER, người dùng có thể bấm nhiều hướng khác nhau rồi, và nếu không lưu hết các phím hợp lệ và hàng đợi thì đến lúc xử lý thông điệp WM\_TIMER có thể sinh ra lỗi rắn chạy sai hướng.

* 1. *Xử lý khi bấm nút LÀM MỚI*

Chỉ làm mới trạng thái của game khi rắn đang không chạy (playing == FALSE). Đầu tiên là thiết đặt lại các thông số của chương trình như kích thước rắn (size = 3), điểm số (score = 0), biến đánh dấu rắn chạy (playing = FALSE), hướng di chuyển (direction = VK\_RIGHT), kích thước của hàng đợi hướng di chuyển (count\_queue = 0).

Sau đó khởi tạo lại vị trí của rắn và tạo mồi mới rồi vẽ lại vùng Client để hiển thị trạng thái mới của chương trình.

* 1. *Xử lý thông điệp WM\_TIMER*

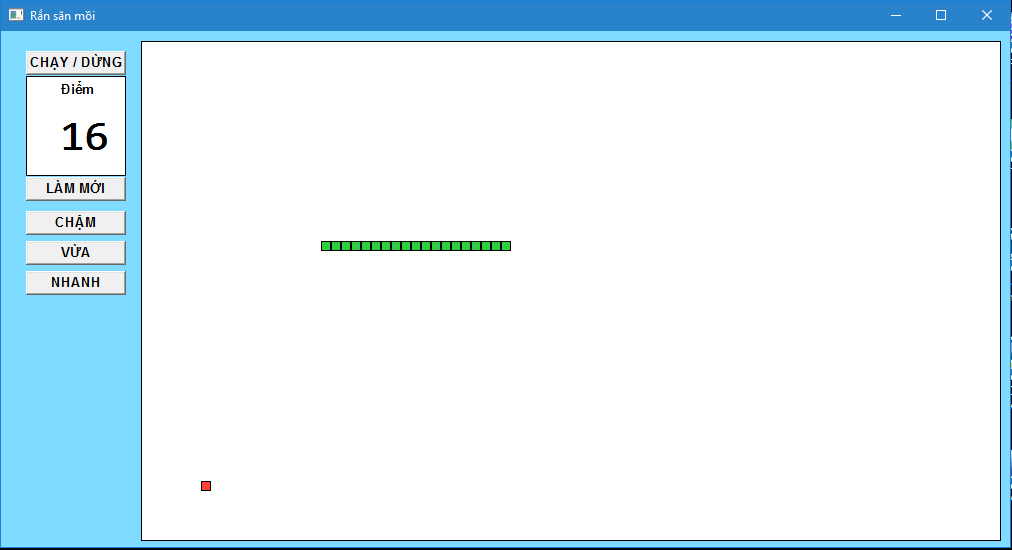
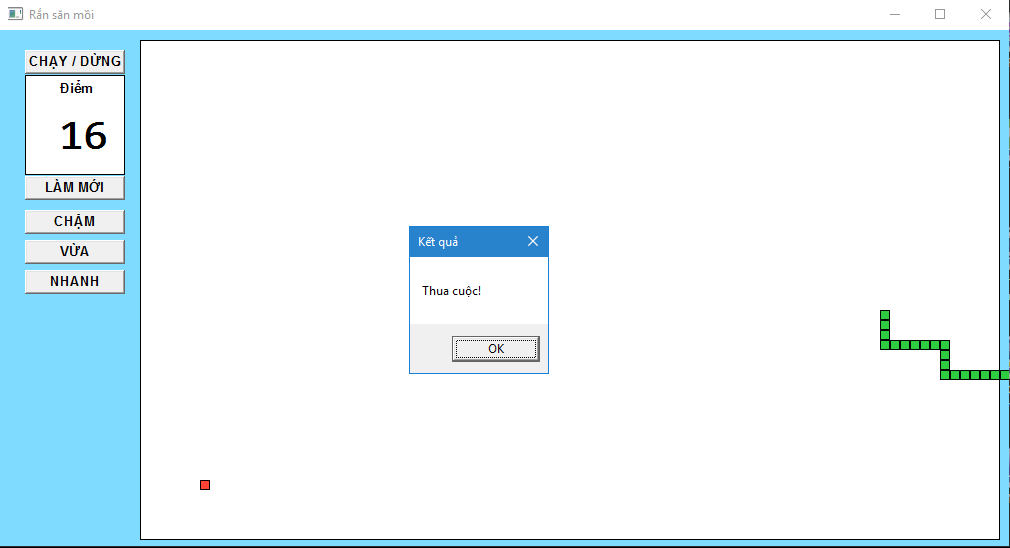
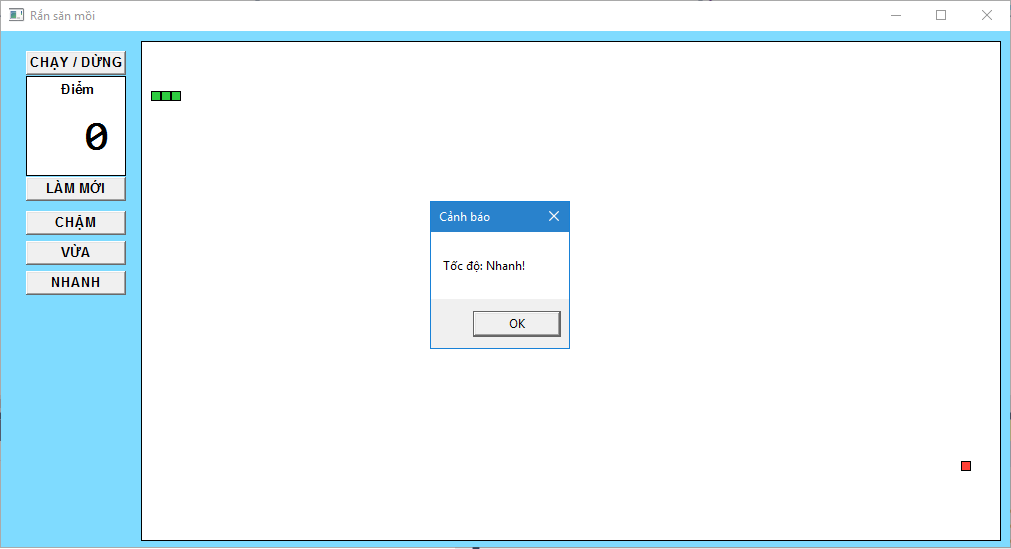
Đầu tiên chúng ta kiểm tra xem rắn có “chết” hay không như đã trình bày ở mục *f*. Nếu rắn "chết" thì ngừng timer và đưa ra thông báo rồi thoát ra khỏi thủ tục.

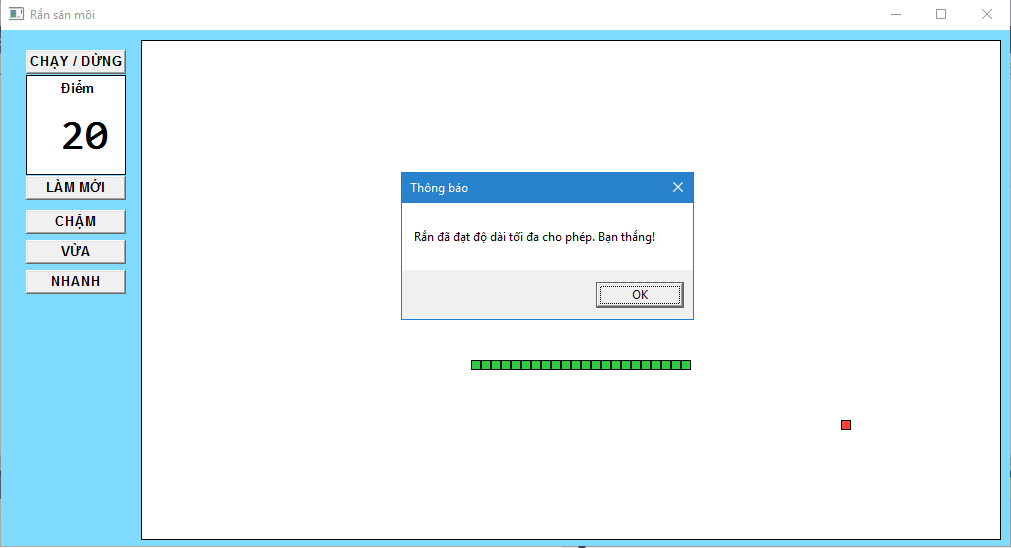
Ngược lại, ta kiểm tra xem rắn có ăn mồi hay không (đầu rắn trùng với con mồi). Nếu có thì tăng kích thước rắn (size++), sau đó lùi các phần tử của rắn lại 1 đơn vị rồi gán đầu rắn là mồi vừa ăn, tăng điểm lên (score++) và vẽ lại vùng hiện điểm, vẽ lại vùng hiển thị cũ của mồi để xoá nó đi, tạo mồi mới rồi vẽ lại vùng mồi mới hiển thị để hiện ra chương trình.

Tiếp theo, kiểm tra nếu rắn đạt độ dài tối đa cho phép thì ngừng timer và đưa ra thông báo, rồi thoát thủ tục.

Nếu rắn vẫn đang ở độ dài cho phép thì di chuyển rắn bằng cách vẽ lại vùng đuối rắn để xoá đuôi rắn ở vị trí cũ đi, rồi dịch các phần tủ của rắn bằng giá trị của phần tử trước nó. Sau đó tính toạ độ mới của đầu rắn và vẽ lại vùng đầu rắn để dịch chuyển.

1. **Một số hình ảnh**

* ****Khi chạy chương trình.
* Khi thay đổi tốc độ
* Khi rắn chết
* Khi độ dài rắn đạt mức tối đa

1. **Kết luận và hướng phát triển**
2. **Kết luận**

Chương trình đã chạy được đúng như thiết kế mong muốn.

1. **Hướng phát triển**

* Cải thiện giao diện đẹp hơn.
* Cải thiện rắn di chuyển mượt hơn
* Cải thiện chức năng di chuyển, chưa thật sự tốt. Đôi khi vẫn mắc lỗi không bắt được hướng di chuyển.