TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM
MÔN: **KĨ THUẬT LẬP TRÌNH**

HƯỚNG DẪN ĐỒ ÁN

RẮN SĂN MỒI

TP.HCM, ngày 20 tháng 02 năm 2017

MỤC LỤC

1 Giới thiệu

Trong phần đồ án này ta sẽ phối hợp các kĩ thuật và cấu trúc dữ liệu cơ bản để xây dựng một trò chơi đơn giản, rắn săn mồi.

Để thực hiện được đồ án này ta cần các kiến thức cơ bản như: xử lý tập tin, tiểu trình, handle, cấu trúc dữ liệu mảng một chiều…

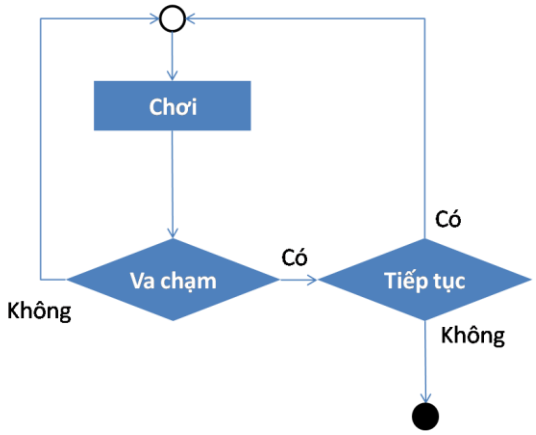
Phần hướng dẫn giúp sinh viên xây dựng trò chơi ở mức độ cơ bản, các em tự nghiên cứu để hoàn thiện một cách tốt nhất có thể.

2 Kịch bản trò chơi

Lúc đầu khi vào game sẽ xuất hiện một "snake" và sẽ tự động di chuyển, người chơi sử dụng các phím 'W', 'A', 'S', 'D' để điều chỉnh hướng di chuyển.

Khi "snake" va chạm tường thì chương trình thông báo yêu cầu người chơi chọn phím 'y' nếu muốn tiếp tục (chương trình sẽ reset trò chơi lại như lúc ban đầu) hoặc chọn 'bất kì phím nào' nếu muốn thoát trò chơi.

Khi "snake" ăn được 4 "food" thì tốc độ di chuyển nhanh hơn, nghĩa là lên 1 cấp (độ dài của snake sẽ trở về kích thước lúc đầu là 6). Khi lên cấp 3 thì dữ liệu sẽ reset lại như lúc ban đầu.



Hình 1: Sơ đồ kịch bản trò chơi

3 Các bước xây dựng trò chơi

Trong phần này ta sẽ lần lượt đi qua các bước xây dựng trò chơi. Lưu ý đây chỉ là một gợi ý lập trình, sinh viên có thể tự thiết kế mẫu phù hợp trong quá trình làm đồ án.

Bước 1: Trong bước này ta sẽ cố định màn hình với kích thước thích hợp, làm điều này giúp tránh trường hợp người dùng tự co giãn màn hình sẽ gây khó khăn trong quá trình xử lý.

Dòng	
1	`void FixConsoleWindow() {`
2	`    HWND consoleWindow = GetConsoleWindow();`
3	`    LONG style = GetWindowLong(consoleWindow, GWL_STYLE);`
4	`    style = style & ~(WS_MAXIMIZEBOX) & ~(WS_THICKFRAME);`
5	`    SetWindowLong(consoleWindow, GWL_STYLE, style);`
6	`}`

Trong đoạn mã trên, kiểu HWND là một con trỏ trỏ tới chính cửa sổ Console. Để làm việc với các đối tượng đồ họa này, ta cần có những kiểu như thế. Cờ GWL_STYLE được xem là dấu hiệu để hàm GetWindowLong lấy các đặc tính mà cửa sổ Console đang có. Kết quả trả về của hàm GetWindowLong là một số kiểu long, ta sẽ hiệu chỉnh tại dòng số 4. Ý nghĩa là để làm mờ đi nút maximize và không cho người dùng thay đổi kích thước cửa sổ hiện hành. Sauk hi đã hiệu chỉnh xong, ta dùng hàm SetWindowLong để gán kết quả hiệu chỉnh trở lại. Ta có thể thử nghiệm hàm trên và tự xem kết quả.

Bước 2: Trong trò chơi sẽ có rất nhiều vị trí mà ta muốn in tại đó, vì vậy ta cần có khả năng di chuyển tới tất cả các vị trí trong màn hình console.

Dòng	
1	`void GotoXY(int x, int y) {`
2	`    COORD coord;`
3	`    coord.X = x;`
4	`    coord.Y = y;`
5	`    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);`
6	`}`

Trong đoạn mã này ta sử dụng struct _COORD (COORD), đây là một cấu trúc dành xử lý cho tọa độ trên màn hình console. Ta gán hoành độ và tung độ cho biến coord sau đó thiết lập vị trí lên màn hình bằng hàm SetConsoleCursorPosition. Lưu ý: hàm này cần một đối tượng chính là màn hình console (màn hình đen), vì vậy ta cũng cần có một con trỏ trỏ tới đối tượng này (HANDLE thực chất là void*). Ta có được bằng cách gọi hàm GetStdHandle với tham số là một cờ STD_OUTPUT_HANDLE.

Bước 3: Tiếp theo ta cần có dữ liệu để phục vụ trò chơi, để đơn giản ta sử dụng biến toàn cục. Ý nghĩa của từng biến và hằng số sinh viên tự xem trong bảng.

Dòng	
1	`//Hằng số`

```
2  #define MAX_SIZE_SNAKE 10
3  #define MAX_SIZE_FOOD 4
4  #define MAX_SPEED 3
5  //Biến toàn cục
6  POINT snake[10]; //Con rắn
7  POINT food[4]; // Thức ăn
8  int CHAR_LOCK;//Biến xác định hướng không thể di chuyển (Ở một thời điểm có một hướng rắn không thể di chuyển)
9  int MOVING;//Biến xác định hướng di chuyển của snake (Ở một thời điểm có ba hướng rắn có thể di chuyển)
10 int SPEED;// Có thể hiểu như level, level càng cao thì tốc độ càng nhanh
11 int HEIGH_CONSOLE, WIDTH_CONSOLE;// Độ rộng và độ cao của màn hình console
12 int FOOD_INDEX; // Chỉ số food hiện hành đang có trên màn hình
13 int SIZE_SNAKE; // Kích thước của snake, lúc đầu có 6 và tối đa lên tời 10
14 int STATE; // Trạng thái sống hay chết của rắn
```

Bước 4: Bước tiếp theo ta sẽ xây dựng hàm ResetData, mục tiêu của hàm này là để thiết lập dữ liệu về trạng thái ban đầu. Đặc biệt trong hàm này có gọi hàm con là GenerateFood, hàm này giúp khởi tạo giá trị vị trí cho các thức ăn, dĩ nhiên tọa độ của mảng food phải khác với tọa độ của snake, vì vậy trong hàm GenerateFood có gọi hàm IsValid để kiểm tra vị trí tọa độ.

```
Dòng
1  bool IsValid(int x, int y) {
2     for (int i = 0; i < SIZE_SNAKE; i++) {
3        if (snake[i].x == x && snake[i].y == y) {
4           return false;
5     return true;
6  }
7
8  void GenerateFood() {
9     int x, y;
10    srand(time(NULL));
11    for (int i = 0; i < MAX_SIZE_FOOD; i++) {
12       do {
13          x = rand() % (WIDTH_CONSOLE -1) + 1;
14          y = rand() % (HEIGH_CONSOLE -1) + 1;
15       } while (IsValid(x, y) == false);
16       food[i] = { x,y };
17    }
18 }
19
```

```
20  void ResetData() {
21     //Khởi tạo các giá trị toàn cục
22     CHAR_LOCK = 'A', MOVING = 'D', SPEED = 1; FOOD_INDEX = 0, WIDTH_CONSOLE = 70,
    HEIGH_CONSOLE = 20, SIZE_SNAKE = 6;
23  //Khởi tạo giá trị mặc định cho snake
24     snake[0] = { 10, 5 }; snake[1] = { 11, 5 };
25     snake[2] = { 12, 5 }; snake[3] = { 13, 5 };
26     snake[4] = { 14, 5 }; snake[5] = { 15, 5 };
27     GenerateFood();//Tạo mảng thức ăn food
28  }
```

Ta sẽ bắt đầu giải thích tại hàm IsValid, hàm này sẽ kiểm tra tọa độ đầu vào có trùng với bất kì một phần tử nào trong mảng snake hay không. Nếu có thì hàm trả về false báo hiệu tọa độ này không hợp lệ. Ngược lại sẽ trả về true.

Với hàm GenerateFood, hàm này sẽ tạo ngẫu nhiên các phần tử cho mảng food. Ở đây ta sử dụng kĩ thuật tạo bộ số ngẫu nhiên. Hàm srand(time(NULL)) (Lưu ý: ta có thể để trong hàm main()) sẽ tạo một hạt giống để tạo bộ số ngẫu nhiên, hàm time(NULL) trả ra số giây (kiểu long) tính từ ngày 1 tháng 1 năm 1970 tới nay. Trong vòng lặp do while ta thấy sẽ tạo ngẫu nhiên liên tục cho các phần tử mảng food (mỗi phần từ mảng này có kiểu POINT). Nếu tọa độ hợp lệ thì sẽ gán cho phần tử này food[i] = {x, y}

Cuối cùng là hàm ResetData(), trong hàm này ta sẽ gán các giá trị khởi đầu cho trò chơi.

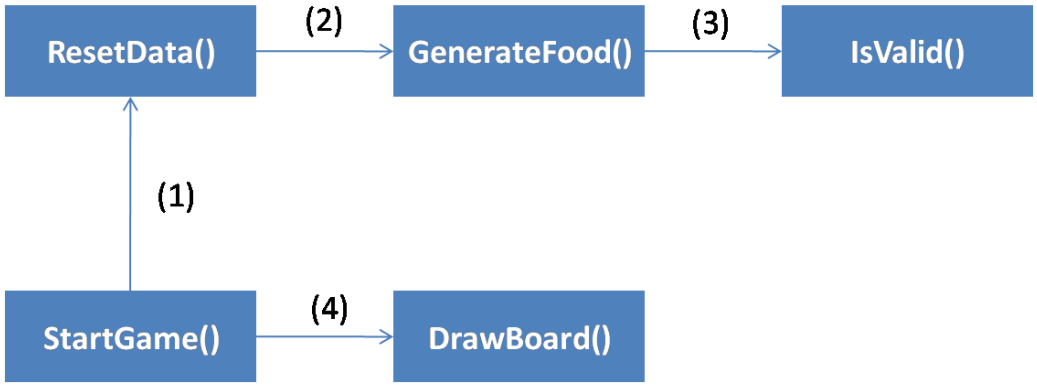
Hình 2: Sơ đồ gọi hàm từ ResetData()

Bước 5: Tiếp theo ta sẽ xây dựng hàm StartGame(), hàm này thực chất là tập các công việc cần làm trước khi vào trò chơi

```
Dòng
1  void StartGame() {
2     system("cls");
3     ResetData(); // Khởi tạo dữ liệu gốc
4     DrawBoard(0, 0, WIDTH_CONSOLE, HEIGH_CONSOLE); // Vẽ màn hình game
5     STATE = 1;//Bắt đầu cho Thread chạy
6  }
7
8  void DrawBoard(int x, int y, int width, int height, int curPosX = 0, int curPosY
    = 0){
9     GotoXY(x, y);cout << 'X';
```

```
10    for (int i = 1; i < width; i++)cout << 'X';
11    cout << 'X';
12    GotoXY(x, height + y);cout << 'X';
13    for (int i = 1; i < width; i++)cout << 'X';
14    cout << 'X';
15    for (int i = y + 1; i < height + y; i++){
16       GotoXY(x, i);cout << 'X';
17       GotoXY(x + width, i);cout << 'X';
18    }
19    GotoXY(curPosX, curPosY);
20 }
```

Dòng mã đầu tiên để xóa trắng màn hình, dòng mã thứ hai là lời gọi hàm ResetData() nhằm khôi phục dữ liệu mặc định ban đầu. Kế đó là gọi hàm DrawBoard() để vẽ hình chữ nhật xung quanh "snake". Dòng lệnh cuối cùng là quan trọng, khi STATE = 1 thì đối tượng "snake" mới chính thức được vẽ ra màn hình.



Hình 3: Sơ đồ gọi hàm từ StartGame()

Sinh viên tự xem xét hàm DrawBoard()

Bước 4: Ngoài hàm StartGame(), ta cần xây dựng hai hàm ExitGame() và PauseGame() nhằm thực hiện chức năng thoát và dừng trò chơi khi cần.

```
Dòng
1  //Hàm thoát game
2  void ExitGame(HANDLE t) {
3     system("cls");
4     TerminateThread(t, 0);
6  }
7
8  //Hàm dừng game
```

```
9   void PauseGame(HANDLE t) {
10     SuspendThread(t);
11  }
```

Trong hàm ExitGame() ta thực hiện xóa trắng màn hình và ngắt tiểu trình đang chạy, ta cần có HANDLE của tiểu trình cần ngắt. Lưu ý: Do tính chất minh họa đơn giản nên sử dụng hàm ngắt trực tiếp một cách rất thô bạo, nếu tiểu trình có chứa các biến con trỏ thì việc ngắt như vậy sẽ làm rò rĩ bộ nhớ.

Trong hàm PauseGame() ta thực hiện tạm dừng chương trình bằng hàm SuspendThread()

Bước 5: Tiếp theo ta xây dựng hàm xử lý khi "snake" ăn "food". Khi đó chiều dài của "snake" sẽ dài ra thêm một phần tử, và phần tử food tiếp theo trong mảng food sẽ được vẽ ra. Do mảng **food đã được tạo ra trước đó nên có khả năng tọa độ của các phần tử food sẽ vô tình khớp với các phần tử của "snake"**.

```
Dòng
1   //Hàm cập nhật dữ liệu toàn cục
2   void Eat() {
3      snake[SIZE_SNAKE] = food[FOOD_INDEX];
4       if (FOOD_INDEX == MAX_SIZE_FOOD - 1)
5       {
6           FOOD_INDEX = 0;
7           SIZE_SNAKE = 6;
8           if (SPEED == MAX_SPEED) SPEED = 1;
9           else SPEED++;
10          GenerateFood();
11      }
12      else {
13          FOOD_INDEX++;
14          SIZE_SNAKE++;
15      }
16  }
```

Nếu tất cả các phần tử food đã được snake ăn hết, vậy ta cần thiết lập các biến toàn cục để tăng tốc độ di chuyển của rắn.. Ngược lại nếu vẫn chưa ăn hết mảng food thì ta cho vẽ phần tử food tiếp theo cũng như tăng kich thước của snake lên.

Hình 4: Sơ đồ gọi hàm từ Eat()

Bước 6: Hàm xử lý khi snake va chạm với tường rào xung quanh. Lúc này mình chuyển trạng thái cho snake là STATE = 0 nhằm để tạm dừng vòng lặp vẽ snake, và sẽ in ra dòng chữ báo hiệu người dùng có muốn tiếp tục (phím 'y') hoặc thoát trò chơi (phím bất kì)

Dòng	
1	`//Hàm xử lý khi snake chết`
2	`void ProcessDead() {`
3	`   STATE = 0;`
4	`   GotoXY(0, HEIGH_CONSOLE + 2);`
5	`   printf("Dead, type y to continue or anykey to exit");`
6	`}`

Bước 7: Hàm DrawSnakeAndFood() nhằm mục đích vẽ phần tử food và snake ra màn hình một cách liên tục. Hàm này đơn giản chỉ duyệt mảng snake sau đó in ra các kí tự một cách liên tục.

Dòng	
1	`//Hàm vẽ màn hình`
2	`void DrawSnakeAndFood(char* str){`
3	`   GotoXY(food[FOOD_INDEX].x, food[FOOD_INDEX].y);`
4	`   printf(str);`
5	`   for (int i = 0; i < SIZE_SNAKE; i++){`
6	`      GotoXY(snake[i].x, snake[i].y);`
7	`      printf(str);`
8	`   }`
9	`}`

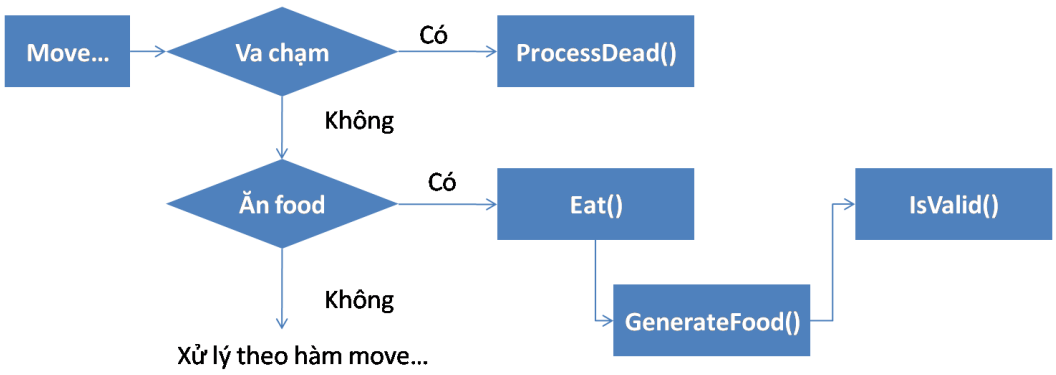
Bước 8: Tiếp theo ta xây dựng các hàm Up, Down, Left, Right để xử lý các hướng di chuyển của snake. Ta thấy khi di chuyển ta cần kiểm tra xem snake có va chạm với tường rào hay không (If đầu tiên). Nếu có va chạm ta gọi hàm ProcessDead() và thoát ra ngoài vì lúc này không cần cập nhật mảng snake. Nhưng nếu không va chạm mà snake ăn food (if thứ nhất trong else) thì ta gọi hàm Eat(). Sau khi ăn food xong thì ta thực hiện cập nhật các phần tử của snake tiến lên một bước (thực chất là lấy giá trị phần tử sau gán cho phần tử trước đó).

Dòng	
1	`void MoveRight() {`
2	`   if (snake[SIZE_SNAKE - 1].x + 1 == WIDTH_CONSOLE) {`
3	`      ProcessDead();`
4	`   }`
5	`   else {`
6	`      if (snake[SIZE_SNAKE - 1].x + 1 == food[FOOD_INDEX].x && snake[SIZE_SNAKE - 1].y == food[FOOD_INDEX].y) {`

```
7          Eat();
8       }
9       for (int i = 0; i < SIZE_SNAKE - 1; i++) {
10         snake[i].x = snake[i + 1].x;
11         snake[i].y = snake[i + 1].y;
12      }
13      snake[SIZE_SNAKE - 1].x++;
14   }
15}
16
17void MoveLeft() {
18   if (snake[SIZE_SNAKE - 1].x - 1 == 0) {
19      ProcessDead();
20   }
21   else {
22      if (snake[SIZE_SNAKE - 1].x - 1 == food[FOOD_INDEX].x && snake[SIZE_SNAKE
- 1].y == food[FOOD_INDEX].y) {
23         Eat();
24      }
25      for (int i = 0; i < SIZE_SNAKE - 1; i++) {
26         snake[i].x = snake[i + 1].x;
27         snake[i].y = snake[i + 1].y;
28      }
29      snake[SIZE_SNAKE - 1].x--;
30   }
31}
32
33void MoveDown() {
34   if (snake[SIZE_SNAKE - 1].y + 1 == HEIGH_CONSOLE) {
35      ProcessDead();
36   }
37   else {
38      if (snake[SIZE_SNAKE - 1].x == food[FOOD_INDEX].x && snake[SIZE_SNAKE -
1].y + 1 == food[FOOD_INDEX].y) {
39         Eat();
40      }
41      for (int i = 0; i < SIZE_SNAKE - 1; i++) {
42         snake[i].x = snake[i + 1].x;
43         snake[i].y = snake[i + 1].y;
44      }
45      snake[SIZE_SNAKE - 1].y++;
46   }
```

```
47  }
48
49  void MoveUp() {
50     if (snake[SIZE_SNAKE - 1].y - 1 == 0) {
51        ProcessDead();
52     }
53     else {
54        if (snake[SIZE_SNAKE - 1].x == food[FOOD_INDEX].x && snake[SIZE_SNAKE -
   1].y - 1 == food[FOOD_INDEX].y) {
55           Eat();
56        }
57        for (int i = 0; i < SIZE_SNAKE - 1; i++) {
58           snake[i].x = snake[i + 1].x;
59           snake[i].y = snake[i + 1].y;
60        }
61        snake[SIZE_SNAKE - 1].y--;
62     }
63  }
```

Bên dưới là sơ đồ gọi hàm



Hình 5: Sơ đồ gọi hàm từ Move…()

Bước 9: Tiếp theo ta sẽ xây dựng một thủ tục chạy liên tục cho tiểu trình ta sẽ start trong hàm main().

```
Dòng
1   //Thủ tục cho thread
2   void ThreadFunc() {
3      while(1) {
4         if (STATE == 1) {//Nếu vẫn còn snake vẫn còn sống
5            DrawSnakeAndFood(" ");
6            switch (MOVING){
7               case 'A':
```

```
8                 MoveLeft();
9                 break;
10              case 'D':
11                MoveRight();
12                break;
13              case 'W':
14                MoveUp();
15                break;
16              case 'S':
17                MoveDown();
18                break;
19          }
20          DrawSnakeAndFood("O");
21          Sleep(1000 / SPEED);//Hàm ngủ theo tốc độ SPEEED
22      }
23   }
24 }
```

Trong thủ tục này ta thấy có vòng lặp vô tận, nếu STATE == 1 tức là snake đang sống thì ta mới tiếp tục kiểm tra biến MOVING xử lý di chuyển, ngược lại STATE == 0 có nghĩa là rắn đã chết như vậy vòng lặp sẽ không làm gì (Màn hình khi đó sẽ giữ nguyên nội dung). Hàm DrawSnakeAndFood(“ ”) nhằm xóa vị trí trước đó của snake để hàm DrawSnakeAndFood(“0”) vẽ lại snake tại vị trí mới. Sau đó cho tiểu trình ngủ khoảng (1000/SPEED). SPEED càng lớn thì tiểu trình ngủ càng ít và snake di chuyển sẽ nhanh hơn. Trong hàm này ta dùng switch để kiểm tra phím người dùng bấm di chuyển để gọi hàm cho thích hợp.

Bước 10: Cuối cùng ta sẽ xây dựng hàm main kết nối tất cả các hàm vừa xây dựng.

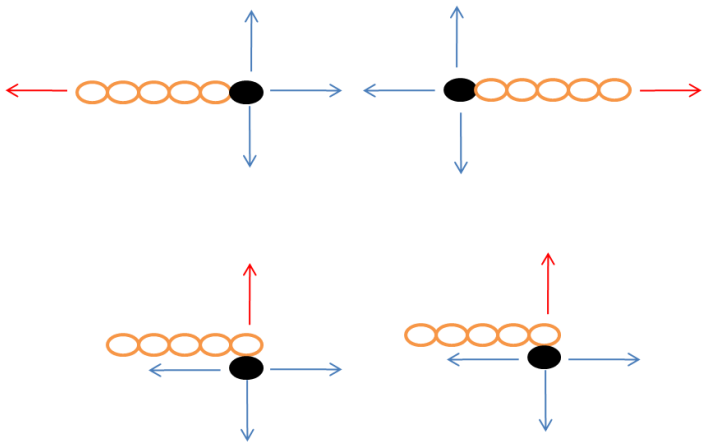
```
Dòng
1  //Hàm main
2  void main(){
3     int temp;
4     FixConsoleWindow();
5     StartGame();
6     thread t1(ThreadFunc); //Tao thread cho snake
7     HANDLE handle_t1 = t1.native_handle(); //Lay handle cua thread
8     while(1){
9        temp = toupper(getch());
10       if (STATE == 1) {
11          if (temp == 'P'){
12             PauseGame(handle_t1);
```

```
13          }
14          else if (temp == 27) {
15             ExitGame(handle_t1);
16             return;
17          }
18          else{
19             ResumeThread(handle_t1);
20             if ((temp != CHAR_LOCK) && (temp == 'D' || temp == 'A' || temp == 'W' || temp == 'S'))
21             {
22                if (temp == 'D') CHAR_LOCK = 'A';
23                else if (temp == 'W') CHAR_LOCK = 'S';
24                else if (temp == 'S') CHAR_LOCK = 'W';
25                else CHAR_LOCK = 'D';
26                MOVING = temp;
27             }
28          }
29       }
30       else {
31          if (temp == 'Y') StartGame();
32          else {
33             ExitGame(handle_t1);
34             return;
35          }
36       }
37    }
38 }
```

Công việc đầu tiên của hàm main() là cố định kích thước và làm mờ nút phóng to. Sau đó gọi hàm StartGame() để chuẩn bị một số thứ cần thiết như dữ liệu ban đầu cho trò chơi. Tiếp theo hàm main() sẽ tạo ra một tiểu trình chạy song song với nó. Thủ tục của tiểu trình này chính là hàm ThreadFunc() ta đã xây dựng trong bước 9. Do có nhu cầu dừng tiểu trình cũng như hủy tiểu trình nên ra cần lấy HANDLE của tiểu trình dành cho xử lý trong hàm PauseGame() và ExitGame().

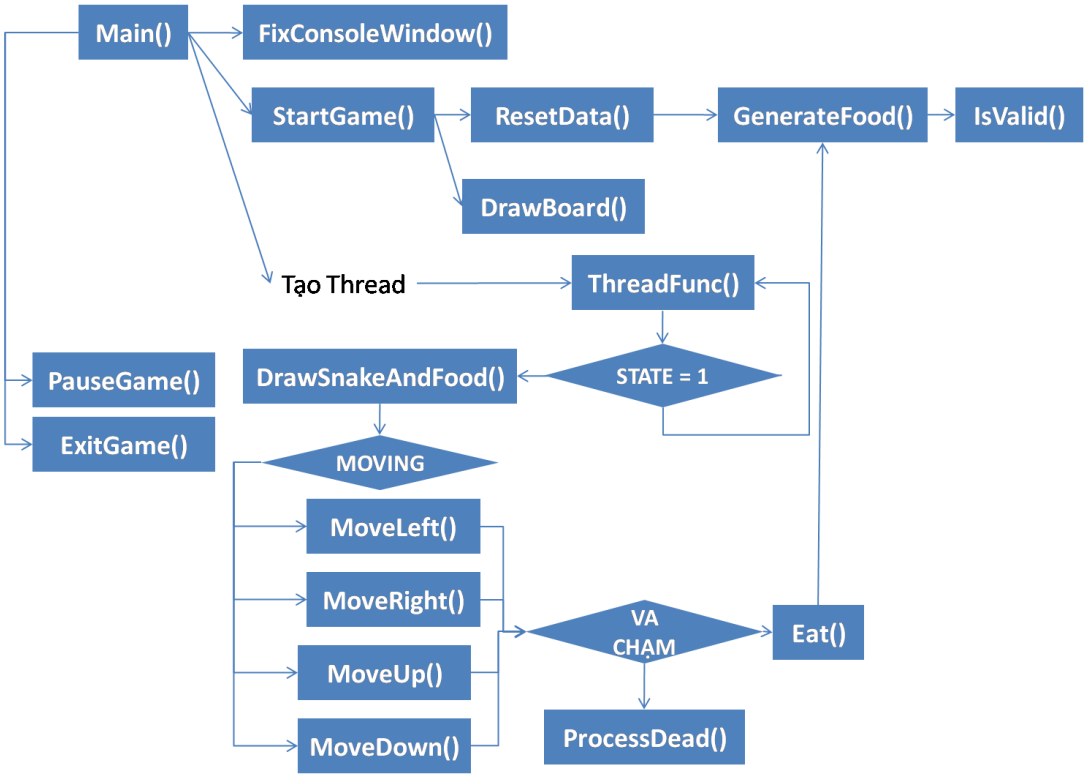
Ban đầu vòng lặp ta sẽ cho người dùng nhập phím. Sau đó tùy vào trạng thái sống (STATE = 1) hay chết (STATE = 0) để xử lý thích hợp. Nếu snake chết thì ta sẽ hỏi người dùng có muốn tiếp tục hay không, nếu chọn ‘y’ thì ta gọi lại StartGame() chơi lại từ đầu, ngược lại ta gọi ExitGame() để thoát. Trường hợp snake còn sống thì ta kiểm tra xem người dùng có nhấn phím ‘P’ và ‘ESC’ hay không. Nếu có thì gọi hàm PauseGame() hoặc ExitGame() tương ứng. Trường hợp người dùng nhấn phím di chuyển thì ta xem

hướng đầu snake phía nào thì ta khóa phía còn lại tương ứng trong biến CHAR_LOCK (tại một thời điểm có một hướng snake không thể di chuyển).



Hình 6: Hướng di chuyển của snake

Để dễ theo dõi hàm main() ta xem sơ đồ gọi hàm bên dưới



Hình 7: Sơ đồ gọi hàm từ main()

4 YÊU CẦU ĐỒ ÁN

Trong phần hướng dẫn trên ta còn thiếu một vài chức năng cơ bản

4.1 Xử lý khi đầu snake chạm vào thân snake (2đ)

Trong hướng dẫn chưa xử lý việc đầu snake chạm vào thân snake. Khi điều này xảy ra ta cũng xử lý tạm dừng trò chơi và hỏi ý kiến người dùng xem có muốn tiếp tục hay không?

4.2 Xử lý lưu trò chơi/tải trò chơi đã lưu (2đ)

Sinh viên bổ sung chức năng khi người dùng nhấn phím 'L' thì xuất hiện dòng lệnh yêu cầu người dùng nhập tên tập tin cần lưu lại. Tương tự khi nhấn phím 'T' thì xuất hiện dòng lệnh yêu cầu người dùng nhập tên tập tin cần tải lên.

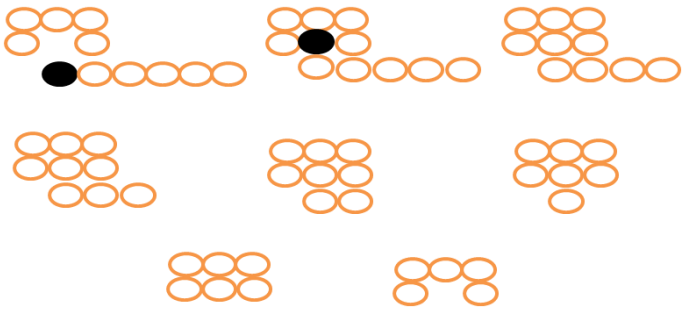
Hướng dẫn: Sinh viên tự tổ chức cấu trúc tập tin để lưu dữ liệu biến toàn cục trong chương trình

4.3 Xử lý giữ nguyên độ dài snake (2đ)

Trong hướng dẫn, mỗi khi lên cấp (SPEED++) thì kích thước snake được thiết lập lại từ đầu (mặc định là 6). Sinh viên xử lý sao cho độ dài của snake vẫn phải giữ nguyên khi lên cấp. Chỉ khi lên MAX_SPEED thì mới thiết lập lại từ đầu.

4.4 Xử lý khi ăn xong food ở một cấp (2đ)

Trong hướng dẫn, khi snake ăn hết 4 phần tử trong mảng food thì sẽ lên cấp. Tuy nhiên sinh viên hiệu chỉnh lại sao cho sau khi ăn 4 phần tử mảng food thì sẽ xuất hiện một cổng cho snake quay về. Khi snake quay về xong thì mới xử lý lên cấp.



4.5 Xử lý hiệu ứng khi va chạm (1đ)

Khi snake va chạm với tường rào, thân snake hoặc cổng vào thì tạo hiệu ứng đơn giản minh họa việc va chạm.

4.6 Xử lý màn hình chính (1đ)

Khi mở trò chơi, chương trình cho phép người dùng chọn ‘Tải lại’ (phím ‘T’) hoặc ‘Bắt đầu chơi’ (phím bất kì). Nếu người dùng chọn phím ‘T’ thì yêu cầu người dùng nhập tên tập tin và sau đó vào trò chơi. Ngược lại thì bắt đầu trò chơi với dữ liệu gốc mặc định.

Snake màn hình chính là chuỗi mã số sinh viên, ví dụ sinh viên có mã số là 1312918 thì chuỗi snake là: 1312918, khi snake ăn mồi thì chèn kí tự đầu tiên lặp lại. Ví dụ sau khi snake ăn mồi thứ nhất thì chuỗi snake là 13129181, ăn mồi lần thứ hai sẽ là 131291813…