ĐẠI HỌC KHOA HỌC TỰ NHIÊN - ĐHQGHCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO
ĐỒ ÁN KĨ THUẬT LẬP TRÌNH

ĐỀ TÀI

SNAKE GAME

GV hướng dẫn: TS. Trương Toàn Thịnh

Lớp 21CTT4
Sinh viên thực hiện (Nhóm 10):

21120410 - Nguyễn Tuấn Anh
21120426 - Huỳnh Phát Đạt
21120430 - Ngô Tuấn Đạt
21120442 - Trần Đăng Duy
21120495 - Phạm Thị Mỹ Lệ

TP. Hồ Chí Minh, ngày 13 tháng 5 năm 2022

Lời mở đầu

Đây là bản báo cáo đồ án lập trình game Rắn săn mồi thuộc khuôn khổ đồ án môn học Kĩ thuật lập trình. Mục tiêu của bản báo cáo trình bày quá trình nghiên cứu, xây dựng game Rắn săn mồi bằng ngôn ngữ C/C++. Qua đó hiểu được cách xây dựng game bằng đồ họa console, cách áp dụng những kiến thức cơ bản của lập trình như *đọc/ghi file, mảng, xử lý chuỗi,...* cũng như những kiến thức khác liên quan đến *xử lý đồ họa, xử lý âm thanh, đa luồng,...*

Để có thể hoàn thành đồ án, nhóm chúng tôi đã tìm tòi, tham khảo từ nhiều nguồn tài liệu khác nhau, cùng với đó là sự giúp đỡ của bạn bè và thầy cô. Xin gửi lời cảm ơn chân thành đến GV.Trương Toàn Thịnh đã tận tình truyền đạt kiến thức trong quá trình học tập môn Kĩ thuật lập trình. Cảm ơn những người bạn đã nhiệt tình giúp đỡ, để đồ án của chùng tôi được hoàn chỉnh hơn.

Nội dung của bản báo cáo bao gồm 3 phần chính, bao gồm cả phần code bằng ngôn ngữ C/C++ khi xây dựng chương trình:

- **Chương 1:** Tổng quan về đồ án, xác định, phân tích các yêu cầu, sơ lược cài đặt chương trình
- **Chương 2:** Chi tiết cài đặt chương trình
- **Chương 3:** Các kết quả khi chạy chương trình

Mặc dù đã rất nỗ lực trong quá trình thực hiện đồ án, tuy nhiên sản phẩm của chúng tôi không thể tránh khỏi những thiếu sót. Chúng tôi rất mong được nhận những sự góp ý để có thể hoàn thiện hơn trong tương lai.

Mục lục

CHƯƠNG 1

Tổng quan

1.1 Giới thiệu

Snake Game hay Rắn săn mồi là một tựa game đã xuất hiện từ lâu, dựa trên trò chơi arcade của Gremlin Industries[1] với cách chơi đơn giản - di chuyển một con rắn (biểu thị bằng những ô vuông xếp liên tiếp nhau) xung quanh một khu vực chơi nhất định đến vị trí thức ăn (những chấm nhỏ xuất hiện ngẫu nhiên trên màn hình), độ dài của rắn phụ thuộc vào số lượng mồi mà nó ăn được. Trò chơi kết thúc khi rắn chạm vào tường, chướng ngại vật được thiết kế sẵn hoặc chạm vào thân của chính nó. Luật chơi rất đơn giản nhưng cũng đầy thử thách.

1.2 Yêu cầu

1.2.1 Yêu cầu tin học

- Ngôn ngữ: C/C++.
- Phương pháp: Lập trình hướng thủ tục (POP).
- IDE: Visual Studio 2019 - 2022.

1.2.2 Yêu cầu giao diện

- Thiết kế giao diện mở đầu game.
- Thiết kế Menu để người chơi có thể dễ dàng lựa chọn các chức năng khác nhau.
- Giao diện dễ nhìn, thân thiện với người chơi, hiển thị đầy đủ thông tin như trạng thái, tốc độ, màn chơi, số điểm,...

1.2.3 Yêu cầu chức năng

- Xử lý trong trường hợp đầu của con rắn chạm vật cản hoặc chạm vào thân của chính nó.
- Save/Load: Save để lưu thông tin của người chơi vào file, với yêu cầu nhập tên người chơi. Load để tải thông tin của màn chơi mà người chơi đã lưu.

[1] một nhà sản xuất trò chơi arcade của Mỹ hoạt động từ năm 1971 đến năm 1983 có trụ sở tại San Diego, California, Hoa Kỳ

- Giữ nguyên độ dài của rắn sau khi qua màn chơi mới. Độ dài của rắn sẽ được reset lại khi quay lại màn chơi đầu.
- Tại mỗi màn chơi, khi con rắn ăn đủ 4 thức ăn thì sẽ xuất hiện một cánh cổng. Nếu con rắn hoàn toàn đi qua cổng thì nó sẽ bước sang màn chơi tiếp theo. Ngược lại nếu đụng trúng cổng, rắn sẽ chết.
- Hiệu ứng: Tạo những hiệu ứng sống động khi rắn chết, hoặc có sự kiện đặc biệt xảy ra.
- Xây dựng thân rắn bằng MSSV của các thành viên trong nhóm. Mỗi khi rắn ăn thức ăn thì chữ số tiếp theo của MSSV sẽ được thêm vào cuối thân rắn.

1.3 Cài đặt

1.3.1 Mô tả

Dựa trên yêu cầu đồ án, trò chơi nhóm xây dựng được mô tả như sau:

Khi bắt đầu trò chơi, màn hình giới thiệu trò chơi xuất hiện và người chơi được yêu cầu nhấn một phím bất kỳ từ bàn phím để tiếp tục. Sau màn hình giới thiệu trò chơi là giao diện Menu bao gồm 6 chức năng:

- **Start:** Bắt đầu một màn chơi mới
- **About:** Xem thông tin các thành viên trong nhóm
- **High Score:** Bảng xếp hạng 15 người chơi có diểm cao nhất
- **Load:** Tải màn chơi mà người chơi đã lưu trước đó
- **Help:** Hướng dẫn chơi game
- **Exit:** Thoát khỏi trò chơi

Người chơi sử dụng các phím W, S để di chuyển lên xuống và [SPACE] để chọn các chức năng trong Menu, phím [ESC] để quay lại Menu. Khi người chơi chọn màn chơi mới hoặc tải một màn chơi đã lưu, Giao diện man hình chơi sẽ hiện lên. Người chơi sử dụng 4 phím W, A, S, D để điều khiển con rắn di chuyển theo các hướng lên, trái, xuống, phải tương ứng. Mỗi lần con rắn ăn một thức ăn thì độ dài rắn sẽ tăng lên 1. Khi ăn đủ 4 thức ăn thì sẽ xuất hiên cổng ở vị trí bất kì trên khung. Lúc con rắn hoàn toàn đi qua cổng thì màn chơi tiếp theo bắt đầu. Trò chơi được thiết kế bao gồm 4 level như sau:

- **Level 1:** Chỉ gồm một bức tường bao quanh khu vực di chuyển của rắn.
- **Level 2:** Xây dựng thêm chướng ngại vật mà rắn phải tránh.
- **Level 3:** Xuất hiện một object di chuyển theo nhiều hướng trong khung, nếu rắn đụng trúng object này, rắn sẽ chết.
- **Level 4:** Rắn phải phá các bức tường và đưa bóng vào khung thành để có thể qua màn.

Khi người chơi hoàn thành level 4, độ dài của rắn sẽ được reset lại về như lúc ban đầu, đồng thời tốc độ di chuyển của rắn tăng lên 1 đơn vị. Khi rắn chạm tường, vật thể hoặc chính nó thì rắn sẽ chết. Lúc đó người chơi có thể lưu lại kết quả của mình. Những kết quả đã lưu sẽ được sắp xếp và hiện thị theo thứ tự từ cao đến thấp trong HighScore (chỉ hiển thị tối đa 15 người chơi). Hơn nữa, trong quá trình chơi game, người chơi có thể tạm dừng và lưu kết quả game hiện tại. Kết quả được lưu có thể được sử dụng để tải lại màn chơi mà người chơi chưa hoàn thành.

1.3.2 Sơ lược cài đặt

Từ những mô tả trên, chương trình được cài đặt gồm các nhóm hàm, biến và hằng được khai báo trong các file header (và được định nghĩa trong file .cpp tương ứng) sau:

- ***Common.h*:** Chứa các thư viện cần thiết để thực hiện chương trình, các hằng số sử dụng trong chương trình.
- ***Globalvariable.h*:** Khai báo các biến toàn cục sử dụng trong chương trình.
- ***System.h*:** Bao gồm các hàm liên quan đến màn hình Console.
- ***Sound.h*:** Bao gồm các hàm xử lý âm thanh.
- ***Draw.h*:** Bao gồm các hàm liên quan đến xử lý đồ họa của trò chơi.
- ***Menu.h*:** Bao gồm các hàm liên quan đến Menu và các hàm chức năng của Menu.
- **Snake.h:** Bao gồm các hàm xử lý trạng thái, điều khiển di chuyển của rắn.
- ***Obj.h*:** Bao gồm các hàm xử lý trạng thái của object.
- ***Gameplay.h*:** Bao gồm các hàm khởi tạo, các hàm liên quan đến các thao tác chơi game.
- ***Highscore.h*:** Bao gồm các hàm liên quan đến chức năng highscore của trò chơi.
- ***SaveLoad.h*:** Bao gồm các hàm liên quan đến chức năng Save/Load của trò chơi.

Cuối cùng, file **snake0.cpp** chứa hàm main() gọi các hàm thực thi theo trình tự thực hiện của trò chơi.

CHƯƠNG 2

Chi tiết cài đặt

Cách thức tổ chức, chi tiết cài đặt chương trình được trình bày kèm theo mã nguồn C/C++ như sau:

2.1 Các thư viện và hằng trong file Common.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <Windows.h>
#include <time.h>
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <thread>
#include <fstream>
#include <iomanip>
#include <string>

using namespace std;

// sz snake >= sz food
#define MAX_SIZE_SNAKE 45
#define MAX_SIZE_FOOD 5
#define MAX_SPEED 3
#define SIZE_WALL 52
#define BACK_GROUND_COLOR 15
#define TEXT_COLOR 6
#define MAX_NAME 10
#define MAX_SUPER_FOOD 5
#define MAX_SIZE_SMALL_WALL 28
#define Max 6
#define Min 1
```

Toàn bộ các hằng, thư viện sử dụng trong chương trình được khai báo và định nghĩa trong file này. Trong đó các hằng được định nghĩa có những vai trò và ý nghĩa khác nhau như kích thước mảng cấu tạo nên rắn và thức ăn, tốc độ tối đa của rắn, các màu mặc định,...

2.2 Các biến toàn cục trong file Globalvariable.h

Để xây dựng dữ liệu cho chương trình, các biến toàn cục (*global variables*) được sử dụng để có thể dễ dàng thao tác trong mã nguồn. Vai trò của các biến được mô tả như trong mã nguồn

dưới đây:

```
extern POINT snake[MAX_SIZE_SNAKE]; //snake
extern POINT food[MAX_SIZE_FOOD]; // food
extern POINT gate[5];  //cong qua man moi
extern POINT out[2]; //cong ra khi bat dau man moi
extern POINT wall[SIZE_WALL]; //tuong o level 2
extern POINT object[4];  //vat the o level 3
extern POINT superfood[5];  //sieu thuc an o level 3
extern POINT smallwall[MAX_SIZE_SMALL_WALL]; //cac buc tuong nho trong level 4
extern POINT goal[14]; //khung thanh trong level 4
extern POINT ball[2]; //qua bong trong level 4

extern char mssv[41]; //ma so sinh vien cua cac thanh vien trong nhom
extern int CHAR_LOCK;//used to determine the direction my snake cannot move (At a moment, there is one direction my snake cannot move to)
extern int MOVING;//used to determine the direction my snake moves (At a moment, there are three directions my snake can move)
extern int SPEED;// Standing for level, the higher the level, the quicker the speed
extern int FOOD_INDEX; // current food-index
extern int SIZE_SNAKE; //do dai cua ran theo thoi gian thuc, thay doi khi ran qua cong
extern int SIZE_SNAKE_1;  //do dai cua ran khong doi, dung de phuc hoi do dai cua ran moi khi buoc vao level moi
extern int STATE; // State of snake: dead or alive
extern int GATE;  //trang thai mo hay chua mo cua cong qua man
extern int SCORE; //diem so cua nguoi choi
extern int flag;  //bien co ho tro thao tac tao cong de ran chui ra khi bat dau man choi
extern int outgate;  //bien kiem tra ran ra chui hoan toan ra khoi cong hay chua
extern int fix;  //bien co de va lai cho trong khi ran chui ra khoi cong
extern int fixcount;  //bien dem de ho tro cho viec va lai cho trong khi ran chui ra khoi cong
extern int boardcolor[4]; //mang chua cac gia tri mau de tao hieu ung nhap nhay khung game
extern int colorboard; //bien nhan cac gia tri mau de tao hieu ung nhap nhay khung game
extern int level; //so thu tu man choi
extern int object_state;  //trang thai di chuyen cua vat the object[0]
extern int object_state1;  //trang thai di chuyen cua vat the object[1]
extern int object_state2;  //trang thai di chuyen cua vat the object[2]
extern int object_state3;  //trang thai di chuyen cua vat the object[3]
extern int eated;  //so phan tu cua mang superfood_eated
extern int superfood_eated[MAX_SUPER_FOOD];  //mang chua cac sieu thuc an
extern int stoptime;  //bien kiem tra ran co an sieu thuc an dung thoi gian hay khong
extern int supercount;  //bien dem thoi gian thuc thi chuc nang khi ran an sieu thuc an dung thoi gian
extern int undying;  //bien kiem tra ran co an sieu thuc an bat tu hay khong
extern int supercount1;  //bien dem thoi gian thuc thi chuc nang khi ran an sieu thuc an bat tu
extern int colorundying; //bien nhan cac gia tri mau sac de tao hieu ung cho ran khi bat tu
extern int object_fix; //bien co de tach vat the lan thu nhat
extern int object_fix1; // bien co de tach vat the lan thu hai
extern int play_again;  //bien tang toc do game khi choi lai tu dau
extern int ball_state;  //trang thai di chuyen cua bong o level 4
extern int smallwall_hitted[MAX_SIZE_SMALL_WALL]; //mang chua cac buc tuong
```

```
    nho da bi pha huy o kevek 4
extern int smallwall_hitted_idx; //so phan tu cua mang smallwall_hitted
extern int goalgoalgoal; //bien kiem tra bong da vao khung thanh o level 4
    hay chua
extern int length_break; //bien kiem tra bong co pha tuong hay khong de thuc
     hien cong diem va do dai
extern int Old_smallwall_hitted_idx; //bien luu lai nhung buc tuong da bi
    pha o level 4
extern int goal_celebration; //bien dem de tao hieu ung cho khung thanh o
    level 4


//set ConsoleBoard
extern int WIDTH_CONSOLE; //do dai chieu ngang cua khung game
extern int HEIGH_CONSOLE; //do dai chieu doc cua khung game
extern int cornerX; //
extern int cornerY;

//set LeftBoard
extern int corLx;
extern int corLy;
extern int widthL;
extern int heightL;

//set FrameBoard
extern int corFx;
extern int corFy;
extern int widthF;
extern int heightF;

//set LevelBoard
extern int widthLevel;
extern int heightLevel;
extern int corLevelx;
extern int corLevely;

extern char* name;

extern string nAmE;
extern int exitgAme;

//-------------------------------Sound-------------------------
extern bool eAt;
extern bool dIe;
extern bool nextlv;
extern bool sUperfood1;
extern bool click;
extern bool goalSound;
extern bool countdownsound1;
extern bool countdownsound2;
//-------------------------------------------------------------
extern bool menuOn;
extern int save_load;
```

2.3 Nhóm hàm khai báo trong file System.h

- Hàm `FixConsoleWindow()`: Cố định kích thước màn hình Console

```
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 code trên, HWND là một handle trỏ tới cửa sổ Console. Cờ GWL_STYLE đánh dấu để hàm GetWindowLong lấy các thuộc tính của màn hình Console. Các thuộc tính này được hiệu chỉnh ở dòng 4. Mục tiêu là vô hiệu hóa nút "maximize" trên cửa sổ Console và ngăn người dùng thay đổi kích thước hiện hành. Sau khi đã hiệu chỉnh, hàm SetWindowLong được sử dụng để gắn kích thước đã hiệu chỉnh trở lại.

• Hàm `GotoXY()`: Di chuyển con trỏ đến vị trí có tọa độ (x, y)

```
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 code trên, struct COORD được sử dụng. Đây là môt struct dành cho xử lý tọa độ trên màn hình Console. Gán các giá trị x, y vào biến coord, sau đó sử dụng hàm SetConsoleCursorPosition() để thiết lập vị trí con trỏ trên màn hình Console.

• Hàm `Nocursortype()`: Ẩn con trỏ nháy khỏi màn hình Console

```
1 void Nocursortype()
2 {
3   CONSOLE_CURSOR_INFO Info;
4   Info.bVisible = FALSE;
5   Info.dwSize = 20;
6   SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &Info);
7 }
```

Trong đoạn code trên, struct CONSOLE_CURSOR_INFO được sử dụng. Đây là một struct chứa các thuộc tính về con trỏ của màn hình Console. Khi trường dữ liệu bVisible của biến Info được gán băng FALSE thì con trỏ sẽ ở trạng thái ẩn. Sau đó hàm SetConsoleCursorInfo() để thiết lập con trỏ vừa định nghĩa trên màn hình Console.

• Hàm `DisableSelection()`: Vô hiệu hóa chọn màn hình Console bằng chuột

```
1 void DisableSelection()
2 {
3   HANDLE hStdin = GetStdHandle(STD_INPUT_HANDLE);
4   SetConsoleMode(hStdin, ~ENABLE_QUICK_EDIT_MODE);
5 }
```

• Hàm `TextColor()`: Hàm thay đổi màu kí tự

```
1 void TextColor(WORD color)
2 {
3   HANDLE hConsoleOutput;
4   hConsoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
5   CONSOLE_SCREEN_BUFFER_INFO screen_buffer_info;
6   GetConsoleScreenBufferInfo(hConsoleOutput, &screen_buffer_info);
7   WORD wAttributes = screen_buffer_info.wAttributes;
8   color &= 0x000f;
```

```
  wAttributes &= 0xfff0;
  wAttributes |= color;
  SetConsoleTextAttribute(hConsoleOutput, wAttributes);
}
```

Hàm nhận vào đối số là một biến kiểu sruct WORD. Hàm cho phép thay đổi màu của kí tự. Thư viện Windows.h cung cấp 16 màu, tương ứng với 16 số từ 0 đến 15.

• Hàm `BackGroundColor()`: Hàm thay đổi màu nền của màn hình Console.

```
void BackGroundColor(WORD color)
{
  HANDLE hConsoleOutput;
  hConsoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
  CONSOLE_SCREEN_BUFFER_INFO screen_buffer_info;
  GetConsoleScreenBufferInfo(hConsoleOutput, &screen_buffer_info);
  WORD wAttributes = screen_buffer_info.wAttributes;
  color &= 0x000f;
  color <<= 4;
  wAttributes &= 0xff0f;
  wAttributes |= color;
  SetConsoleTextAttribute(hConsoleOutput, wAttributes);
}
```

Hàm nhận vào đối số là một biến kiểu struct WORD. Hàm cho phép thay đổi màu nền khi thực hiện các thao tác trên màn hình Console.

2.4 Nhóm hàm khai báo trong file Sound.h

• Hàm `Sound()`:

```
void Sound() {
  while (1) {
    if (eAt == 1) {
      PlaySound(TEXT("eatingsound.wav"), NULL, SND_SYNC);
      eAt = 0;
    }
    if (sUperfood1 == 1) {
      PlaySound(TEXT("superfood.wav"), NULL, SND_SYNC);
      sUperfood1 = 0;
    }
    if (dIe == 1) {
      PlaySound(TEXT("gameover.wav"), NULL, SND_SYNC);
      dIe = 0;
    }
    if (nextlv == 1) {
      PlaySound(TEXT("win.wav"), NULL, SND_SYNC);
      nextlv = 0;
    }
    if (goalSound == 1) {
      PlaySound(TEXT("goal.wav"), NULL, SND_SYNC);
      goalSound = 0;
    }
    //click/selectclick: DrawLoadBoard();
    if (countdownsound1 == 1) {
      PlaySound(TEXT("countdown1.wav"), NULL, SND_SYNC);
      countdownsound1 = 0;
    }
    if (countdownsound2 == 1) {
```

```
29        PlaySound(TEXT("countdown2.wav"), NULL, SND_SYNC);
30        countdownsound2 = 0;
31      }
32    }
33  }
```

Hàm Sound() với vòng lặp while chứa các biến toàn cục nằm rải rác trong chương trình, khi cần phát âm thanh, các biến này được gán bằng 1, khi đó hàm Sound() sẽ phát âm thanh tương ứng và trả các biến này về lại 0.

2.5 Nhóm hàm khai báo trong file Draw.h

• Hàm `printASCII()`: Hàm in ký tự ASCII từ file

```
void printASCII(string fileName, int x, int y)
{
  string line = "";
  ifstream inFile;
  inFile.open(fileName);
  if (inFile.is_open()) {
    while (getline(inFile, line)) {
      GotoXY(x, y++);
      cout << line << endl;
    }
  }
  else
    cout << "File failed to load" << endl;
  inFile.close();
}
```

Các kí tự trong file được in tại điểm khởi đầu có tọa độ (x, y). Thông báo hiện ra nếu chương trình không đọc được file. Hàm hỗ trợ xây dựng giao diện game bằng cách in ra mẫu chữ đa dạng lưu trong file text khiến game bắt mắt hơn.

• Hàm `ClearBoard()`: Xóa kí tự bên trong khung game

```
void ClearBoard()
{
  for (int i = cornerX + 1; i < cornerX + WIDTH_CONSOLE; i++)
  {
    for (int j = cornerY + 1; j < cornerY + HEIGH_CONSOLE; j++)
    {
      GotoXY(i, j); cout << " ";
    }
  }
  TextColor(3);
  TextColor(TEXT_COLOR);
}
```

Sử dụng vòng lặp for đi đến từng tọa độ trong khung chơi, thực hiện in ra kí tự ‘ ’ để thay thế cho các kí tự ban đầu.

• Hàm `DrawCountDown()`: Hàm vẽ đếm ngược trước khi bắt đầu màn chơi mới

```
void DrawCountDown(int x, int y)
{
  TextColor(6);
  printASCII("num3.txt", x, y);
```

```
5   countdownsound1 = 1;
6   Sleep(1000);
7
8   printASCII("num2.txt", x, y);
9   countdownsound1 = 1;
10  Sleep(1000);
11
12  printASCII("num1.txt", x, y);
13  countdownsound1 = 1;
14  Sleep(1000);
15
16  printASCII("go.txt", x - 3, y);
17  countdownsound2 = 1;
18  Sleep(1000);
19
20  ClearBoard();
21 }
```

"3", "2", "1", "GO" lần lượt được lưu trong 3 file num3.txt, num2.txt, num1.txt, go.txt , liên tiếp được in ra sau mỗi 1 giây bằng lệnh Sleep(1000) ở dòng 6,10,14.

• `DrawFrame()`: Hàm vẽ khung giới hạn di chuyển của rắn

```
1  void DrawFrame(int x, int y)
2  {
3    TextColor(2);
4    int row = 223, col = 219, topleft = 219, topright = 219, botleft = 223,
       botright = 223;
5    // top
6    GotoXY(x, y); cout << char(topleft);
7    for (int i = 1; i < widthF; i++) {
8      cout << char(row);
9    }
10   cout << char(topright);
11
12   // left & right
13   for (int i = y + 1; i < heightF + y; i++)
14   {
15     GotoXY(x, i); cout << char(col);
16     GotoXY(x + widthF, i); cout << char(col);
17   }
18
19   // bottom
20   GotoXY(x, heightF + y); cout << char(botleft);
21   for (int i = 1; i < widthF; i++) {
22     cout << char(row);
23   }
24   cout << char(botright);
25 }
```

• Hàm `DeleteFrameLevel()`: Áp dụng xóa một phần khung bao quát được vẽ bởi hàm DrawFrame()

```
1  void DeleteFrameLevel(int x, int y)
2  {
3    for (int i = x; i < x + widthLevel; i++)
4    {
5      for (int j = y; j < y + heightLevel; j++)
6      {
7        GotoXY(i, j); cout << " ";
8      }
```

```
  }
}
```

Mục đích của hàm là tạo không gian vẽ tiêu đề Level 1, Level 2, Level 3, Level 4.

• Các hàm `DrawFrameLevel(), DrawFrameLevel1(), DrawFrameLevel2(), DrawFrameLevel3()`: các hàm vẽ khung [widthLevel x heightLevel] và tiêu đề Level (lưu tại các file theo thứ tự "level.txt", "level1.txt", "level2.txt", "level3.txt) tại tọa độ (x,y)

```
void DrawFrameLevel(int x, int y)
{

  TextColor(10);
  int row = 223, col = 219, topleft = 219, topright = 219, botleft = 223,
   botright = 223;
  // top
  GotoXY(x, y); cout << char(topleft);
  for (int i = 1; i < widthLevel; i++) {
    cout << char(row);
  }
  cout << char(topright);

  // left & right
  for (int i = y + 1; i < heightLevel + y; i++)
  {
    GotoXY(x, i); cout << char(col);
    GotoXY(x + widthLevel, i); cout << char(col);
  }

  // bottom
  GotoXY(x, heightLevel + y); cout << char(botleft);
  for (int i = 1; i < widthLevel; i++) {
    cout << char(row);
  }
  cout << char(botright);
  printASCII("level1.txt", x + 11, y + 1);

}
void DrawFrameLevel1(int x, int y)
{
  TextColor(10);
  int row = 223, col = 219, topleft = 219, topright = 219, botleft = 223,
   botright = 223;
  // top
  GotoXY(x, y); cout << char(topleft);
  for (int i = 1; i < widthLevel; i++) {
    cout << char(row);
  }
  cout << char(topright);

  // left & right
  for (int i = y + 1; i < heightLevel + y; i++)
  {
    GotoXY(x, i); cout << char(col);
    GotoXY(x + widthLevel, i); cout << char(col);
  }

  // bottom
  GotoXY(x, heightLevel + y); cout << char(botleft);
  for (int i = 1; i < widthLevel; i++) {
```

```
    cout << char(row);
  }
  cout << char(botright);
  printASCII("level2.txt", x + 11, y + 1);
}

void DrawFrameLevel2(int x, int y)
{
  TextColor(10);
  int row = 223, col = 219, topleft = 219, topright = 219, botleft = 223,
   botright = 223;
  // top
  GotoXY(x, y); cout << char(topleft);
  for (int i = 1; i < widthLevel; i++) {
    cout << char(row);
  }
  cout << char(topright);

  // left & right
  for (int i = y + 1; i < heightLevel + y; i++)
  {
    GotoXY(x, i); cout << char(col);
    GotoXY(x + widthLevel, i); cout << char(col);
  }

  // bottom
  GotoXY(x, heightLevel + y); cout << char(botleft);
  for (int i = 1; i < widthLevel; i++) {
    cout << char(row);
  }
  cout << char(botright);
  printASCII("level3.txt", x + 11, y + 1);
}
void DrawFrameLevel3(int x, int y)
{
  TextColor(10);
  int row = 223, col = 219, topleft = 219, topright = 219, botleft = 223,
   botright = 223;
  // top
  GotoXY(x, y); cout << char(topleft);
  for (int i = 1; i < widthLevel; i++) {
    cout << char(row);
  }
  cout << char(topright);

  // left & right
  for (int i = y + 1; i < heightLevel + y; i++)
  {
    GotoXY(x, i); cout << char(col);
    GotoXY(x + widthLevel, i); cout << char(col);
  }

  // bottom
  GotoXY(x, heightLevel + y); cout << char(botleft);
  for (int i = 1; i < widthLevel; i++) {
    cout << char(row);
  }
  cout << char(botright);
  printASCII("level4.txt", x + 11, y + 1);
}
```

• Hàm `DrawLeftBoard()`: Vẽ khung bên trái của màn hình, đây là khung chứa thông tin màn chơi được in bởi hàm `DrawInfo()`

```
void DrawLeftBoard(int x, int y)
{
  TextColor(10);
  int row = 205, col = 186, topleft = 201, topright = 187, botleft = 200,
   botright = 188;
  // top
  GotoXY(x, y); cout << char(topleft);
  for (int i = 1; i < widthL; i++) {
    cout << char(row);
  }
  cout << char(topright);

  // left & right
  for (int i = y + 1; i < heightL + y; i++)
  {
    GotoXY(x, i); cout << char(col);
    GotoXY(x + widthL, i); cout << char(col);
  }

  // bottom
  GotoXY(x, heightL + y); cout << char(botleft);
  for (int i = 1; i < widthL; i++) {
    cout << char(row);
  }
  cout << char(botright);

  // midle
  TextColor(0);
  for (int i = 0; i < widthL - 1; i++)
  {
    GotoXY(x + 1 + i, y + 5);
    cout << (char)167;
  }
}
```

• Hàm `DrawInfo()`: In thông tin màn chơi gồm: SCORE, LENGTH, SPEED, FOOD ngay tại thời điểm gameplay

```
void DrawInfo(int x, int y)
{
  GotoXY(x + 1, y);       cout << "SCORE :" << right << setw(7) << SCORE;
  GotoXY(x + 1, y + 1); cout << "LENGTH:" << setw(7) << SIZE_SNAKE;
  GotoXY(x + 1, y + 2); cout << "SPEED :" << setw(7) << SPEED;
  GotoXY(x + 1, y + 3); cout << "FOOD  :" << setw(5) << FOOD_INDEX << "/" <<
    MAX_SIZE_FOOD - 1;
  TextColor(2);
  if (FOOD_INDEX >= 1)
  {
    for (int i = 0; i < 5; i++)
    {
      for (int j = 0; j < 14; j++)
      {
        GotoXY(x + 1 + j, y + 23 - i);
        cout << (char)176;
      }
    }
  }
  if (FOOD_INDEX >= 2)
```

```
20    {
21      for (int i = 5; i < 10; i++)
22      {
23        for (int j = 0; j < 14; j++)
24        {
25          GotoXY(x + 1 + j, y + 24 - i);
26          cout << (char)177;
27        }
28      }
29    }
30    if (FOOD_INDEX >= 3)
31    {
32      for (int i = 10; i < 15; i++)
33      {
34        for (int j = 0; j < 14; j++)
35        {
36          GotoXY(x + 1 + j, y + 24 - i);
37          cout << (char)178;
38        }
39      }
40    }
41    if (FOOD_INDEX >= 4)
42    {
43      for (int i = 15; i < 20; i++)
44      {
45        for (int j = 0; j < 14; j++)
46        {
47          GotoXY(x + 1 + j, y + 24 - i);
48          cout << (char)219;
49        }
50      }
51    }
52  }
```

4 lệnh if() ở dòng 8, 19, 30, 41 kiểm tra FOOD_INDEX để vẽ "thanh năng lượng"

- Hàm `DrawBoard()`: Vẽ khung kích thước [width x height] tại tọa độ (x,y)

```
1  void DrawBoard(int x, int y, int width, int height, int curPosX, int curPosY
       )
2  {
3
4    int row = 205, col = 186, topleft = 201, topright = 187, botleft = 200,
      botright = 188;
5    // top
6    GotoXY(x, y); cout << char(topleft);
7    for (int i = 1; i < width; i++) {
8      cout << char(row);
9    }
10   cout << char(topright);
11
12   // left & right
13   for (int i = y + 1; i < height + y; i++)
14   {
15     GotoXY(x, i); cout << char(col);
16     GotoXY(x + width, i); cout << char(col);
17   }
18
19   // bottom
20   GotoXY(x, height + y); cout << char(botleft);
21   for (int i = 1; i < width; i++) {
```

```
    cout << char(row);
  }
  cout << char(botright);

  TextColor(TEXT_COLOR);
}
```

- Hàm `DrawWall()`: Vẽ chướng ngại vật cho Level 2

```
void DrawWall()
{
  GotoXY(wall[0].x, wall[0].y); cout << (char)188;
  GotoXY(wall[1].x, wall[1].y); cout << (char)200;
  GotoXY(wall[2].x, wall[2].y); cout << (char)187;
  GotoXY(wall[3].x, wall[3].y); cout << (char)201;
  for (int i = 4; i < 36; i++)
  {
    if (i % 2 == 0)
    {
      GotoXY(wall[i].x, wall[i].y);
      cout << (char)205;
    }
    else
    {
      GotoXY(wall[i].x, wall[i].y);
      cout << (char)186;
    }
  }
  for (int i = 36; i < 52; i++)
  {
    GotoXY(wall[i].x, wall[i].y);
    cout << (char)205;
  }
}
```

- Hàm `DrawSmallWall()`: Vẽ các tường nhỏ ở Level 4

```
void DrawSmallWall()
{
  for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
  {
    if (i == 6 || i == 13 || i == 20 || i == 27)
    {
      for (int j = 0; j < 8; j++)
      {
        GotoXY(smallwall[i].x + j, smallwall[i].y);
        cout << char(220);
      }

    }
    else
    {
      for (int j = 0; j < 9; j++)
      {
        GotoXY(smallwall[i].x + j, smallwall[i].y);
        cout << char(220);
      }
    }
  }
}
```

- Hàm `DrawBoard1()`: Vẽ khung cho Level 2

```
void DrawBoard1(int x, int y, int width, int height, int curPosX, int
    curPosY)
{
  TextColor(3);
  int row = 205, col = 186, topleft = 201, topright = 187, botleft = 200,
   botright = 188;
  // top
  GotoXY(x, y); cout << char(topleft);
  for (int i = 1; i < width; i++) {
    cout << char(row);
  }
  cout << char(topright);

  // left & right
  for (int i = y + 1; i < height + y; i++)
  {
    GotoXY(x, i); cout << char(col);
    GotoXY(x + width, i); cout << char(col);
  }

  // bottom
  GotoXY(x, height + y); cout << char(botleft);
  for (int i = 1; i < width; i++) {
    cout << char(row);
  }
  cout << char(botright);

  TextColor(TEXT_COLOR);
  GotoXY(curPosX, curPosY);
}
```

- Hàm `DrawBoard2()`: Vẽ khung cho Level 3

```
void DrawBoard2(int x, int y, int width, int height, int curPosX, int
    curPosY)
{
  int row = 205, col = 186, topleft = 201, topright = 187, botleft = 200,
   botright = 188;
  // top
  GotoXY(x, y); cout << char(topleft);
  for (int i = 1; i < width; i++) {
    cout << char(row);
  }
  cout << char(topright);

  // left & right
  for (int i = y + 1; i < height + y; i++)
  {
    GotoXY(x, i); cout << char(col);
    GotoXY(x + width, i); cout << char(col);
  }

  // bottom
  GotoXY(x, height + y); cout << char(botleft);
  for (int i = 1; i < width; i++) {
    cout << char(row);
  }
  cout << char(botright);

  TextColor(TEXT_COLOR);
```

```
26
27    GotoXY(curPosX, curPosY);
28 }
```

• Hàm `DrawGoal()`: Vẽ khung thanh o Level 4

```
1  void DrawGoal()
2  {
3    for (int i = 0; i < 10; i++)
4    {
5      GotoXY(goal[i].x, goal[i].y);
6      cout << char(205);
7    }
8    GotoXY(goal[10].x, goal[10].y); cout << char(201);
9    GotoXY(goal[11].x, goal[11].y); cout << char(186);
10   GotoXY(goal[12].x, goal[12].y); cout << char(187);
11   GotoXY(goal[13].x, goal[13].y); cout << char(186);
12   GotoXY(goal[11].x + 1, goal[12].y + 1); cout << "__________";
13 }
```

• Hàm `DrawBoard3()`: Vẽ giao diện Level 4

```
1  void DrawBoard3(int x, int y, int width, int height, int curPosX, int
       curPosY)
2  {
3    int row = 205, col = 186, topleft = 201, topright = 187, botleft = 200,
       botright = 188;
4    // top
5    GotoXY(x, y); cout << char(topleft);
6    for (int i = 1; i < width; i++) {
7      cout << char(row);
8    }
9    cout << char(topright);
10
11   // left & right
12   for (int i = y + 1; i < height + y; i++)
13   {
14     GotoXY(x, i); cout << char(col);
15     GotoXY(x + width, i); cout << char(col);
16   }
17
18   // bottom
19   GotoXY(x, height + y); cout << char(botleft);
20   for (int i = 1; i < width; i++) {
21     cout << char(row);
22   }
23   cout << char(botright);
24
25   TextColor(TEXT_COLOR);
26
27   DrawSmallWall();
28   DrawGoal();
29   GotoXY(curPosX, curPosY);
30 }
```

• Hàm `DrawPause()`: Vẽ giao diện Pause Game

```
1  void DrawPause()
2  {
3    if (level == 4)
4    {
5      TextColor(3);
```

```
    for (int i = -1; i < 11; i++)
    {
      GotoXY(goal[0].x + i, goal[0].y);
      cout << char(205);
    }
  }
  TextColor(11);

  // print sub table
  GotoXY(cornerX + 10, cornerY + 7); cout << (char)201;
  for (int i = 1; i < 20; i++)
  {
    GotoXY(cornerX + 10 + i, cornerY + 7); cout << (char)205;
  }
  //top of sub
  for (int i = 0; i < 20; i++)
  {
    GotoXY(cornerX + 41 + i, cornerY + 7); cout << (char)205;
  }
  GotoXY(cornerX + 61, cornerY + 7); cout << (char)187;
  // left and right of sub
  for (int i = 1; i < 10; i++)
  {
    GotoXY(cornerX + 10, cornerY + 7 + i); cout << (char)186;
    GotoXY(cornerX + 61, cornerY + 7 + i); cout << (char)186;
  }
  // bot of sub
  for (int i = 1; i < 51; i++)
  {
    GotoXY(cornerX + 10 + i, cornerY + 17); cout << (char)205;
  }

  GotoXY(cornerX + 10, cornerY + 17); cout << (char)200;
  GotoXY(cornerX + 61, cornerY + 17); cout << (char)188;
  TextColor(10);
  GotoXY(cornerX + 14, cornerY + 15); cout << "CONTINUE";
  GotoXY(cornerX + 13, cornerY + 16); cout << "[PRESS P]";
  TextColor(13);
  GotoXY(cornerX + 52, cornerY + 15); cout << "STOP";
  GotoXY(cornerX + 50, cornerY + 16); cout << "[PRESS S]";

  TextColor(9);
  printASCII("pause.txt", cornerX + 20, cornerY + 3);
  TextColor(TEXT_COLOR);
  // print heart
  for (int i = 1; i < 5; i++)
  {
    GotoXY(cornerX + 32, cornerY + 9 + i); cout << (char)3;
  }
  for (int i = 1; i < 6; i++)
  {
    GotoXY(cornerX + 32 + i, cornerY + 9 + i); cout << (char)3;
    GotoXY(cornerX + 32 + i, cornerY + 14 - i); cout << (char)3;
  }
  for (int i = 0; i < 8; i++)
  {
    GotoXY(cornerX + 38, cornerY + 8 + i); cout << (char)3;
  }
  GotoXY(0, 0);
}
```

Trong đoạn code trên, lệnh if kiểm tra level = 4 để xóa Goal.

• Hàm `ResultBoard()`: Vẽ khung kết quả

```
void ResultBoard()
{
  TextColor(4);
  GotoXY(cornerX + 10, cornerY + 7); cout << (char)201;
  for (int i = 1; i < 51; i++)
  {
    GotoXY(cornerX + 10 + i, cornerY + 7); cout << (char)205;
  }
  GotoXY(cornerX + 61, cornerY + 7); cout << (char)187;
  for (int i = 1; i < 10; i++)
  {
    GotoXY(cornerX + 10, cornerY + 7 + i); cout << (char)186;
    GotoXY(cornerX + 61, cornerY + 7 + i); cout << (char)186;
  }
  GotoXY(cornerX + 10, cornerY + 17); cout << (char)200;
  GotoXY(cornerX + 61, cornerY + 17); cout << (char)188;
  for (int i = 1; i < 51; i++)
  {
    GotoXY(cornerX + 10 + i, cornerY + 17); cout << (char)205;
  }
```

• Hàm `Result()`: Hiển thị màn hình kết quả: kết quả màn chơi được hiển thị gồm: Level, Score, Size, Speed.

```
void Result()
{
  TextColor(4);
  DrawBoard(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE);
  TextColor(6);
  DrawBoard(cornerX + 20, cornerY + 5, WIDTH_CONSOLE / 2 - 4, HEIGH_CONSOLE
   / 2 + 2);
  TextColor(12);


  printASCII("result.txt", cornerX + 23, cornerY + 3);

  GotoXY(cornerX + 31, cornerY + 8); cout << "USER INFO";
  GotoXY(cornerX + 31, cornerY + 9); cout << "^^^^^^^^^^";
  GotoXY(cornerX + 15, cornerY + 18); cout << "EXIT";
  GotoXY(cornerX + 13, cornerY + 19); cout << "[PRESS E]";
  GotoXY(cornerX + 52, cornerY + 18); cout << "SAVE";
  GotoXY(cornerX + 50, cornerY + 19); cout << "[PRESS S]";
  TextColor(0);
  GotoXY(cornerX + 26, cornerY + 10); cout << "Level         :";
  GotoXY(cornerX + 26, cornerY + 12); cout << "Score         :";
  GotoXY(cornerX + 26, cornerY + 14); cout << "Snake's Size :";
  GotoXY(cornerX + 26, cornerY + 16); cout << "Snake's Speed:";
  TextColor(0);
  GotoXY(cornerX + 42, cornerY + 10); cout << level;
  GotoXY(cornerX + 42, cornerY + 12); cout << SCORE;
  GotoXY(cornerX + 42, cornerY + 14); cout << SIZE_SNAKE_1;
  GotoXY(cornerX + 42, cornerY + 16); cout << SPEED;
}
```

• Hàm `ClearSave()`: Xóa kí tự bên trong sau khi chọn Save ở màn hình kết quả

```
void ClearSave()
{
```

```
  TextColor(4);
  for (int i = cornerX + 1; i < cornerX + WIDTH_CONSOLE; i++)
  {
    for (int j = cornerY + 1; j < cornerY + 21; j++)
    {
      GotoXY(i, j); cout << " ";
    }
  }
  DrawBoard(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE);
}
```

- Hàm `SaveBoard()`: Hiển thị màn hình Save Game

```
void SaveBoard()
{
  ClearSave();
  TextColor(9);
  printASCII("savegame.txt", cornerX + 6, cornerY + 2);
  TextColor(0);
  GotoXY(cornerX + 44, cornerY + 7); cout << (char)201;
  GotoXY(cornerX + 69, cornerY + 7); cout << (char)187;
  GotoXY(cornerX + 44, cornerY + 17); cout << (char)200;
  GotoXY(cornerX + 69, cornerY + 17); cout << (char)188;
  for (int i = 1; i < 25; i++)
  {
    GotoXY(cornerX + 44 + i, cornerY + 7); cout << (char)205;
    GotoXY(cornerX + 44 + i, cornerY + 17); cout << (char)205;
  }
  for (int i = 1; i < 10; i++)
  {
    GotoXY(cornerX + 44, cornerY + 7 + i); cout << (char)186;
    GotoXY(cornerX + 69, cornerY + 7 + i); cout << (char)186;
  }
  GotoXY(cornerX + 46, cornerY + 9); cout << "       ONLY";
  GotoXY(cornerX + 46, cornerY + 10); cout << "alphanumeric characters";
  GotoXY(cornerX + 46, cornerY + 12); cout << "      A - Z (a - z)   ";
  GotoXY(cornerX + 46, cornerY + 14); cout << "      0-9     ";
  GotoXY(cornerX + 46, cornerY + 16); cout << "[At most 10 characters]";
  GotoXY(cornerX + 15, cornerY + 9); cout << "Enter your name: ";
  GotoXY(cornerX + 10, cornerY + 12); cout << "=>";
  GotoXY(cornerX + 16, cornerY + 12);

  do {
    GotoXY(cornerX + 16, cornerY + 12); cout << "          ";
    GotoXY(cornerX + 16, cornerY + 12);
    cin.clear();
    name = LimInput(MAX_NAME);
  } while (NameCheck(name, "SaveInf.txt"));

  for (int i = cornerX + 6; i < cornerX + 6 + 60; i++)
  {
    for (int j = cornerY + 2; j < cornerY + 2 + 5; j++)
    {
      GotoXY(i, j); cout << " ";
    }

  }
  TextColor(11);
  printASCII("saved.txt", cornerX + 20, cornerY + 2);
  TextColor(0);
  GotoXY(cornerX + 10, cornerY + 12); cout << "Press ANY KEY to  back to
```

```
    MENU";
49  char xx = _getch();
50 }
```

2.6 Nhóm hàm khai báo trong file Menu.h

• Hàm `void MenuGame()`: Hàm thực hiện chức năng trở về menu hoặc tiếp tục chơi hoặc chơi lại từ đầu

```
1  void MenuGame()
2  {
3    save_load = 0;
4    menuOn = 1;
5    int n = 0;
6    while (1) {
7      if (menuOn) {
8        n = Menu();
9        menuOn = 0;
10       LoadGame(n);
11     }
12     if (!menuOn) break;
13   }
14 }
```

Cách thức hoạt động: Dùng vòng lặp while với cờ là biến menuOn được khởi tạo với giá trị bằng 1, trong while, mở hàm Menu(), nếu chọn chức năng chơi tiếp hoặc chơi lại, hàm Menu() sẽ được trả về 1 số và biến menuOn được gán lại bằng 0, khi đó vòng lặp trở lại menu bị hủy.

• Hàm `About()`: Giới thiệu thông tin các thành viên của nhóm

```
1  void About()
2  {
3    system("cls");
4    TextColor(4);
5    printASCII("introduction.txt", 25, 2);
6    int disX = 20;
7    TextColor(0);
8    for (int i = disX; i < disX + 83; i++)
9    {
10     GotoXY(i, 10); cout << char(196);
11   }
12   GotoXY(disX + 30, 10);
13   cout << " PROJECT HUNTING SNAKE ";
14   int y = 13;
15   GotoXY(disX, y++);
16   cout << "UNIVERSITY OF SCIENCE - HCMUS";
17   GotoXY(disX, y++);
18   cout << "Faculty    Information Technology";
19   GotoXY(disX, y++);
20   cout << "Course     Programming Techniques";
21   GotoXY(disX, y++);
22   cout << "Lecturer   Truong Toan Thinh";
23   GotoXY(disX, 17);
24   cout << "Class       21CTT4";
25
26   y = 13;
27   int disX2 = disX + 60;
28
```

```
  GotoXY(disX2, y++);
  cout << "21120430 - NGO TUAN DAT";
  GotoXY(disX2, y++);
  cout << "21120410 - NGUYEN TUAN ANH";
  GotoXY(disX2, y++);
  cout << "21120442 - TRAN DANG DUY";
  GotoXY(disX2, y++);
  cout << "21120495 - PHAM THI MY LE";
  GotoXY(disX2, y++);
  cout << "21120426 - HUYNH PHAT DAT";

  TextColor(8);
  GotoXY(disX + 25, 20);
  cout << "Press ANY KEY to  back to MENU";
  int press = _getch();
  PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
  system("cls");
}
```

• Hàm `Help()`: Cung cấp các thông tin nhằm hướng dẫn cho người chơi chơi game và để thoát khỏi hàm quay trở lại màn hình menu chính, người chơi có thể ấn bất kì nút nào

```
void Help()
{
  system("cls");
  TextColor(12);
  BackGroundColor(11);

  int disX = 17, disY = 11;
  GotoXY(disX + 10, disY++); cout << "  DIRECT SNAKE  ";
  GotoXY(disX + 35, disY++); cout << "                ";
  GotoXY(disX + 35, disY++); cout << "   'W'      " << char(30) << "    ";
  GotoXY(disX + 35, disY++); cout << "   'S'      " << char(31) << "    ";
  GotoXY(disX + 35, disY++); cout << "   'A'      " << char(17) << "    ";
  GotoXY(disX + 35, disY++); cout << "   'D'      " << char(16) << "    ";
  GotoXY(disX + 35, disY++); cout << "   'P'    Pause" << "  ";
  GotoXY(disX + 35, disY++); cout << "   'E'    Exit" << "   ";
  GotoXY(disX + 35, disY++); cout << "                ";
  TextColor(0);
  BackGroundColor(15);
  disY += 2;
  GotoXY(disX + 10, disY++); cout << char(4) << " Help the snake eat enough
   food to pass each level!";
  GotoXY(disX + 10, disY++); cout << char(4) << " Eat 4 foods and go through
    the gate of each level to move to next level.";
  GotoXY(disX + 10, disY++); cout << char(4) << " When you passed 4 level,
   the game will start again from level 1.";
  GotoXY(disX + 10, disY++); cout << "  the speed of the snake will increase
    1 index " << endl;
  GotoXY(disX + 10, disY++); cout << "  and the length of the snake will be
   unchanged. " << endl;
  GotoXY(disX + 10, disY++); cout << char(4) << " Remember, DO NOT let the
   snake hit the walls or bite itself.";
  TextColor(8);
  GotoXY(disX + 10, disY++); cout << "          Press ANY KEY to back to
   MENU ";
  TextColor(12);
  printASCII("help.txt", 48, 3);

  for (int i = disX; i < disX + 83; i++)
  {
```

```
    GotoXY(i, 9); cout << char(196);
  }
  GotoXY(disX + 33, 9);
  cout << " PROJECT HUNTING SNAKE ";

  int press = _getch();
  PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
  system("cls");
}
```

• Hàm `Exit()`: Thoát game

```
void Exit()
{
  system("cls");
  int color = 0;
  while (color < 7)
  {
    TextColor(color++);
    printASCII("goodbye.txt", 30, 4);

    TextColor(12);
    GotoXY(47, 23);

    TextColor(4);
    GotoXY(42, 12); cout << "THANKS FOR PLAYING OUR SNAKE GAME";
    GotoXY(42, 13); cout << "   HOPE YOU ENJOY THE GAME";
    GotoXY(42, 14); cout << " GOODBYE AND SEE YOU LATER!!!";
    TextColor(0);
    for (int i = 40; i < 80; i++)
    {
      GotoXY(i, 17); cout << char(196);
    }
    Sleep(100);
  }
  Sleep(1000);
  system("cls");
  exit(0);
}
```

Hàm Exit() thực hiện chức năng thoát khỏi chương trình khi người chơi muốn thoát game bằng exit(0). Trước khi chương trình kết thúc, màn hình sẽ tạo một số hiệu ứng nhấp nháy cùng với lời cảm ơn.

• Hàm `Menu()`: Hàm menu chính của game, chứa toàn bộ các hàm thành phần theo thứ tự lần lượt là StartGame, About, HighScore, Load Game, Help và Exit

```
int Menu()
{
  system("cls");

  int position = 1;
  int keyPressed = 0;

  int spaceBox = 3;
  int Xbox = 49, Ybox = 10;
  int wBox = 20, hBox = 2;

  string file = "menu.txt";

  while (true)
```

```
15  {
16    TextColor(3);
17    BackGroundColor(15);
18    printASCII(file, 13, 0);
19
20    boxstart(Xbox, Ybox, wBox, hBox, "START", position);
21
22    boxabout(Xbox, Ybox += spaceBox, wBox, hBox, "ABOUT", position);
23
24    boxhighscore(Xbox, Ybox += spaceBox, wBox, hBox, "HIGH SCORE", position)
      ;
25
26    boxloadgame(Xbox, Ybox += spaceBox, wBox, hBox, "LOAD GAME", position);
27
28
29    boxhelp(Xbox, Ybox += spaceBox, wBox, hBox, "HELP", position);
30
31
32    boxexit(Xbox, Ybox += spaceBox, wBox, hBox, "EXIT", position);
33
34    Ybox = 10;
35
36    keyPressed = _getch();
37
38    if (keyPressed == 115)
39    {
40      if (position == Max)
41      {
42        position = Min;
43      }
44      else
45      {
46        position++;
47      }
48    }
49    else if (keyPressed == 119)
50    {
51      if (position == Min)
52      {
53        position = Max;
54      }
55      else
56      {
57        position--;
58      }
59    }
60
61
62
63    if (keyPressed == 32)
64    {
65      PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
66      switch (position)
67      {
68      case 1: //new game
69        return 0;
70      case 2:
71        About();
72        break;
73      case 3:
```

```
74          DrawScoreBoard(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE);
75          break;
76        case 4: // continue
77          return 1;
78        case 5:
79          Help();
80          break;
81        case 6:
82          Exit();
83          break;
84        }
85      }
86    }
87    return -1;
88  }
```

Ở 2 câu điều kiện "keyPressed == 115" và "keyPressed == 119" sẽ giúp cho người dùng di chuyển lên xuống bằng 'w' và 's' trên bàn phím. Các câu điều kiện nhỏ nằm trong 2 câu điều kiện trên sẽ giúp cho người trên di chuyển lên xuống như đã nói trên và thêm nữa là nếu như biến "position" đang ở mục "Start" thì nếu như người chơi chọn ấn 'w' thì position sẽ trỏ xuống mục "Exit" và ngược lại, nếu biến position đang ở mục "Exit" thì nếu người chơi ấn "s" thì position sẽ tự động trỏ lên mục "Start".

Tiếp theo sẽ là câu lệnh "switch(position)" để người chơi có thể chọn các mục mình muốn trong menu ở trong các trường hợp 1 đến 6 (case:) bằng cách chọn nút 'SPACE' trên bàn phím.

2.7 Nhóm hàm khai báo trong file Snake.h

• Hàm `DrawSnake()`: Xóa rắn cũ sau mỗi vòng lặp

```
1  void DrawSnake(char* str) {
2    for (int i = SIZE_SNAKE - 1; i >= 0; i--) {
3      GotoXY(snake[i].x, snake[i].y);
4      cout << str;
5    }
6  }
```

Vòng lặp for duyệt toàn bộ và đi đến tọa độ thân rắn, in ra str. Với tham số đầu vào str = " ", hàm được sử dụng để xóa con rắn ở vị trí hiện tại, chuẩn bị cho việc vẽ rắn khi nó thay đổi vị trí.

• Hàm `DrawSnakeAndFood()`: Vẽ rắn và food mới sau mỗi vòng lặp

```
1  void DrawSnakeAndFood(char* str) {
2    if (GATE == 0 && level != 4)
3    {
4      TextColor(2);
5      int x = 254;
6      GotoXY(food[FOOD_INDEX].x, food[FOOD_INDEX].y);
7      cout << (char)x;
8      TextColor(TEXT_COLOR);
9    }
10   int m = SIZE_SNAKE_1;
11   int n = m - SIZE_SNAKE;
12   TextColor(TEXT_COLOR);
13   for (int i = SIZE_SNAKE - 1; i >= 0; i--) {
14     if (n == 0)
15     {
```

```
16        TextColor(5);
17      }
18      if (undying)
19      {
20        if (colorundying % 7 == 0 || colorundying % 15 == 0)
21        {
22          colorundying += 2;
23        }
24        TextColor(colorundying++);
25      }
26      if (n == 40)
27      {
28        n = 0;
29      }
30      GotoXY(snake[i].x, snake[i].y);
31      cout << *(str + n++);
32      TextColor(TEXT_COLOR);
33    }
34  }
```

Mỗi khi di chuyển đến tọa đọ mới, rắn và food cũ bị xóa đi bằng hàm DrawSnake() với tham số str = " ", tiếp tục, rắn và food mới được vẽ ra, lệnh if dòng 2 sẽ kiểm tra nếu level = 4 thì không vẽ food. Vòng for dòng 13 in rắn mới.Lệnh if dòng 19 được sử dụng để tạo hiệu ứng đồ họa khi rắn ăn được undying food ở level 3.

• Hàm `Eat()`: Thay đổi các biến cần thiết khi rắn ăn food

```
1  void Eat()
2  {
3    eAt = 1;
4    snake[SIZE_SNAKE] = food[FOOD_INDEX];
5    if (FOOD_INDEX == MAX_SIZE_FOOD - 2)
6    {
7      FOOD_INDEX++;
8      GATE = 1;
9      SCORE++;
10   }
11   else {
12     FOOD_INDEX++;
13     SIZE_SNAKE++;
14     SIZE_SNAKE_1++;
15     SCORE++;
16   }
17 }
```

Hàm "Eat" sẽ thực hiện các thao tác khi rắn ăn thức ăn, độ dài của rắn sẽ tăng thêm 1, điểm số tăng thêm 1 và thức ăn tiếp theo trong chuỗi "food" sẽ xuất hiện. Khi rắn đã ăn đủ thức ăn, biến GATE sẽ được gán bằng 1 và cổng qua level tiếp theo sẽ xuất hiện. Biến "eAt" được gán bằng 1 để phục vụ cho phần âm thanh.

• Hàm `Dead()`: Tao hiệu ứng nhấp nháy cho rắn khi chết

```
1  void Dead(char* str)
2  {
3    for (int i = 0; i < 5; i++)
4    {
5      int m = SIZE_SNAKE_1;
6      int n = m - SIZE_SNAKE;
7      TextColor(12);
8      GotoXY(snake[SIZE_SNAKE - 1].x, snake[SIZE_SNAKE - 1].y);
```

```
    cout << (char)35;
    n++;
    for (int j = SIZE_SNAKE - 2; j >= 0; j--) {
      GotoXY(snake[j].x, snake[j].y);
      cout << *(str + n++);
    }
    Sleep(250);
    TextColor(TEXT_COLOR);
    for (int j = SIZE_SNAKE - 1; j >= 0; j--) {
      GotoXY(snake[j].x, snake[j].y);
      cout << " ";
    }
    Sleep(250);
  }
  TextColor(TEXT_COLOR);
}
```

Hàm "Dead" thực hiện vòng lặp với các thao tác lần lượt là vẽ rắn và xóa rắn xen kẽ nhau bởi các hàm "Sleep". Biến "dIe được gán bằng 1 để phục vụ cho phần âm thanh.

• Hàm `ProcessDead()`: Thực hiện các thao tác khi rắn chết

```
void ProcessDead()
{
  STATE = 0;
  dIe = 1;
  Dead(mssv);
}

```

Hàm "ProcessDead" thực hiện các thao tác khi rắn chết, ngoài việc gán biến STATE = 0 để dừng vòng lặp của game, ta sẽ gọi thêm hàm "Dead" để tạo hiệu ứng nhấp nháy.

• Các hàm điều khiển di chuyển của rắn

```
void MoveRight() {
  if (GATE == 1)
  {
    for (int i = 1; i < 5; i++)
    {
      if (snake[SIZE_SNAKE - 1].x == gate[i].x && snake[SIZE_SNAKE - 1].y ==
     gate[i].y)
      {
        ProcessDead();
        return;
      }
    }
  }
  for (int i = 0; i < SIZE_WALL; i++)
  {
    if (snake[SIZE_SNAKE - 1].x == wall[i].x && snake[SIZE_SNAKE - 1].y ==
    wall[i].y)
    {
      ProcessDead();
      return;
    }
  }
  for (int i = 0; i < SIZE_SNAKE - 1; i++)
  {
    if (snake[SIZE_SNAKE - 1].x + 1 == snake[i].x && snake[SIZE_SNAKE - 1].y
    == snake[i].y)
```

```
    {
      ProcessDead();
    }
  }
  if ((snake[SIZE_SNAKE - 1].x == WIDTH_CONSOLE + cornerX) && snake[
   SIZE_SNAKE - 1].y != gate[0].y) {
    ProcessDead();
  }
  else {
    if (snake[SIZE_SNAKE - 1].x + 1 == food[FOOD_INDEX].x && snake[
   SIZE_SNAKE - 1].y == food[FOOD_INDEX].y) {
      Eat();
    }
    for (int i = 0; i < SIZE_SNAKE - 1; i++) {
      snake[i].x = snake[i + 1].x;
      snake[i].y = snake[i + 1].y;
    }
    snake[SIZE_SNAKE - 1].x++;
  }
}

void MoveLeft() {
  if (GATE == 1)
  {
    for (int i = 1; i < 5; i++)
    {
      if (snake[SIZE_SNAKE - 1].x == gate[i].x && snake[SIZE_SNAKE - 1].y ==
    gate[i].y)
      {
        ProcessDead();
        return;
      }
    }
  }
  for (int i = 0; i < SIZE_WALL; i++)
  {
    if (snake[SIZE_SNAKE - 1].x == wall[i].x && snake[SIZE_SNAKE - 1].y ==
   wall[i].y)
    {
      ProcessDead();
      return;
    }
  }
  for (int i = 0; i < SIZE_SNAKE - 1; i++)
  {
    if (snake[SIZE_SNAKE - 1].x - 1 == snake[i].x && snake[SIZE_SNAKE - 1].y
    == snake[i].y)
    {
      ProcessDead();
    }
  }
  if (snake[SIZE_SNAKE - 1].x == cornerX && snake[SIZE_SNAKE - 1].y != gate[
   0].y) {
    ProcessDead();
  }
  else {
    if (snake[SIZE_SNAKE - 1].x - 1 == food[FOOD_INDEX].x && snake[
   SIZE_SNAKE - 1].y == food[FOOD_INDEX].y) {
      Eat();
    }
```

```
77      for (int i = 0; i < SIZE_SNAKE - 1; i++) {
78        snake[i].x = snake[i + 1].x;
79        snake[i].y = snake[i + 1].y;
80      }
81      snake[SIZE_SNAKE - 1].x--;
82    }
83  }
84
85  void MoveDown() {
86    if (GATE == 1)
87    {
88      for (int i = 1; i < 5; i++)
89      {
90        if (snake[SIZE_SNAKE - 1].y == gate[i].y && snake[SIZE_SNAKE - 1].x ==
      gate[i].x)
91        {
92          ProcessDead();
93          return;
94        }
95      }
96    }
97    for (int i = 0; i < SIZE_WALL; i++)
98    {
99      if (snake[SIZE_SNAKE - 1].y == wall[i].y && snake[SIZE_SNAKE - 1].x ==
     wall[i].x)
100     {
101       ProcessDead();
102       return;
103     }
104   }
105   for (int i = 0; i < SIZE_SNAKE - 1; i++)
106   {
107     if (snake[SIZE_SNAKE - 1].x == snake[i].x && snake[SIZE_SNAKE - 1].y + 1
      == snake[i].y)
108     {
109       ProcessDead();
110     }
111   }
112   if (snake[SIZE_SNAKE - 1].y == HEIGH_CONSOLE + cornerY && snake[SIZE_SNAKE
      - 1].x != gate[0].x) {
113     ProcessDead();
114   }
115   else {
116     if (snake[SIZE_SNAKE - 1].x == food[FOOD_INDEX].x && snake[SIZE_SNAKE -
     1].y + 1 == food[FOOD_INDEX].y) {
117       Eat();
118     }
119     for (int i = 0; i < SIZE_SNAKE - 1; i++) {
120       snake[i].x = snake[i + 1].x;
121       snake[i].y = snake[i + 1].y;
122     }
123     snake[SIZE_SNAKE - 1].y++;
124   }
125 }
126
127 void MoveUp() {
128   if (GATE == 1)
129   {
130     for (int i = 1; i < 5; i++)
131     {
```

```
132       if (snake[SIZE_SNAKE - 1].y == gate[i].y && snake[SIZE_SNAKE - 1].x ==
      gate[i].x)
133       {
134         ProcessDead();
135         return;
136       }
137     }
138   }
139   for (int i = 0; i < SIZE_WALL; i++)
140   {
141     if (snake[SIZE_SNAKE - 1].y == wall[i].y && snake[SIZE_SNAKE - 1].x ==
     wall[i].x)
142     {
143       ProcessDead();
144       return;
145     }
146   }
147   for (int i = 0; i < SIZE_SNAKE - 1; i++)
148   {
149     if (snake[SIZE_SNAKE - 1].x == snake[i].x && snake[SIZE_SNAKE - 1].y - 1
      == snake[i].y)
150     {
151       ProcessDead();
152     }
153   }
154   if (snake[SIZE_SNAKE - 1].y == cornerY && snake[SIZE_SNAKE - 1].x != gate[
     0].x) {
155     ProcessDead();
156   }
157   else {
158     if (snake[SIZE_SNAKE - 1].x == food[FOOD_INDEX].x && snake[SIZE_SNAKE -
     1].y - 1 == food[FOOD_INDEX].y) {
159       Eat();
160     }
161     for (int i = 0; i < SIZE_SNAKE - 1; i++) {
162       snake[i].x = snake[i + 1].x;
163       snake[i].y = snake[i + 1].y;
164     }
165     snake[SIZE_SNAKE - 1].y--;
166   }
167 }
168
```

Các hàm "MoveUp", "MoveLeft", "MoveDown" và "MoveUp" thực hiện các thao tác khi rắn di chuyển 1 trong 4 hướng. Đầu tiên, nếu cổng đã được mở, ta kiểm tra rắn có chạm vào thành của cổng hay không, nếu có thì ta gọi hàm "ProcessDead" và trả về. Tiếp theo ta kiểm tra rắn có chạm vào tường (ở level 2) hay không, nếu có ta cũng gọi hàm "ProcessDead" và trả về. Tiếp đến ta kiểm tra rắn có chạm vào thân hay không, nếu có ta gọi hàm "ProcessDead". Cuối cùng ta kiểm tra rắn có chạm vào tường (xung quanh) hay không, nếu có ta gọi hàm "ProcessDead". Nếu không có điều kiện nào ở trên thỏa mãn và nếu rắn ăn thức ăn, ta gọi hàm "Eat" và điều chỉnh lại vị trí các phần tử của rắn (theo từng trường hợp cho 4 hướng).

- Hàm `GoThrough()`: Khung game nhấp nháy khi rắn đi qua cổng

```
1  void GoThrough()
2 {
3   TextColor(boardcolor[colorboard++]);
4   DrawBoard(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE);
5   DrawGate();
```

```
  if (colorboard == 4)
  {
    colorboard = 0;
  }
  TextColor(TEXT_COLOR);
}

```

• Hàm `GetIn()`: Kiểm tra rắn đã chui qua cổng hay chưa

```
bool GetIn()
{
  for (int i = 0; i < SIZE_SNAKE; i++)
  {
    if (snake[i].x == gate[0].x && snake[i].y == gate[0].y)
    {
      nextlv = 1;
      return true;
    }
  }
  return false;
}

```

Hàm kiểm tra từng phần tử của rắn nếu trùng với đầu vào của cổng (gate[0]) thì trả về true và trả về false nếu ngược lại, biến nextlv được gán giá trị phục vụ cho phần âm thanh.

• Hàm `Process()`: Các thao tác sau khi ăn đủ food

```
void Process()
{
  if (GATE == 1)
  {
    DrawGate();
    if (GetIn())
    {
      GoThrough();
      SIZE_SNAKE--;
      if (level == 3)
      {
        stoptime = 1;
        undying = 0;
      }
    }
    if (SIZE_SNAKE == 0)
    {
      if (level == 1)
      {
        StartGame1();
      }
      else if (level == 2)
      {
        StartGame2();
      }
      else if (level == 3)
      {
        StartGame3();
      }
      else
      {
        play_again++;
```

```
33          int scOre = SCORE;
34          ResetGlobal();
35          StartGame();
36          SCORE = scOre;
37        }
38      }
39    }
40  }
41
```

Hàm "Process" thực hiện các thao tác để qua level mới khi hoàn thành level hiện tại. Đầu tiên ta kiểm tra nếu biến GATE = 1 ta gọi hàm "DrawGate" để vẽ cổng. Sau đó, ta dùng hàm "GateIn" để kiểm tra rắn có đang chui vào cổng hay không. Khi điều kiện rắn đang chui vào cổng được thỏa mãn, ta gọi hàm "GoThrough" để tạo hiệu ứng cho khung game. Đồng thời ta thực hiện "SIZE_SNAKE–" để rắn chui từ từ vào cổng (ở đây ta xét 1 điều kiện riêng cho level 3, "stoptime = 1" để dừng vật thể và undying = 0 để hiệu ứng bất tử không tiếp tục ở level 4). Tiếp theo ta xét trường hợp nếu SIZE_SNAKE = 0 tức là rắn đã chui hoàn toàn vào cổng, ta xét xem level hiện tại là bao nhiêu để gọi hàm "StartGame" tương ứng. Riêng đối với level 4, ta thực hiện "play_again++" để tăng tốc độ cho vòng lặp chơi lại từ level 1, t gọi hàm "ResetGlobal" (được định nghĩa trong Gameplay.cpp) để khởi tạo lại một số biến toàn cục đã thay đổi sau một quá trình chơi. Cuối cùng ta gọi hàm "StartGame" để lặp lại quá trình chơi game (thực hiện scOre = SCORE trước hàm này và SCORE = scOre sau khi gọi hàm để giữ lại điểm số cũ và tiếp tục tích lũy).

2.8 Nhóm hàm khai báo trong file Obj.h

2.8.1 Nhóm hàm xử lý thức ăn (food)

• Hàm `IsValid()`: Kiểm tra nếu vị trí của thức ăn được khởi tạo trùng với các vị trí thành phần của con rắn thì sẽ trả về false, nếu không sẽ trả về true.

```
bool IsValid(int x, int y)
{
  for (int i = 0; i < SIZE_SNAKE; i++)
  {
    if (snake[i].x == x && snake[i].y == y)
    {
      return false;
    }
  }
  return true;
}
```

• Hàm `IsValid1()`: Kiểm tra tọa độ của food có trùng với tọa độ của rắn và tường (level 2) hay không

```
bool IsValid1(int x, int y)
{
  for (int i = 0; i < SIZE_SNAKE; i++)
  {
    if (snake[i].x == x && snake[i].y == y)
    {
      return false;
    }
  }
  for (int i = 0; i < SIZE_WALL; i++)
  {
```

```
    if (wall[i].x == x && wall[i].y == y)
    {
      return false;
    }
  }
  return true;
}
```

• Hàm `IsSuperValid()`: Kiểm tra tọa độ của superfood (level 3) có trung với tọa độ food hay không

```

bool IsSuperValid(int x, int y)
{
  for (int i = 0; i < MAX_SIZE_FOOD; i++)
  {
    if (food[i].x == x && food[i].y == y)
    {
      return false;
    }
  }
  return true;
}
```

• Các hàm khởi tạo thức ăn:

```
void GenerateFood()
{
  int x, y;
  srand(time(NULL));
  for (int i = 0; i < MAX_SIZE_FOOD - 1; i++) {
    do {
      x = rand() % (WIDTH_CONSOLE - 2) + (cornerX + 2);
      y = rand() % (HEIGH_CONSOLE - 2) + (cornerY + 2);
    } while (IsValid(x, y) == false);
    food[i] = { x,y };
  }
}

void GenerateFood1()
{
  int x, y;
  srand(time(NULL));
  for (int i = 0; i < MAX_SIZE_FOOD; i++) {
    do {
      x = rand() % (WIDTH_CONSOLE - 2) + (cornerX + 2);
      y = rand() % (HEIGH_CONSOLE - 2) + (cornerY + 2);
    } while (IsValid1(x, y) == false);
    food[i] = { x,y };
  }
}

void GenerateSuperFood()
{
  int x, y;
  srand(time(NULL));
  for (int i = 0; i < MAX_SUPER_FOOD; i++) {
    do {
      x = rand() % (WIDTH_CONSOLE - 2) + (cornerX + 2);
      y = rand() % (HEIGH_CONSOLE - 2) + (cornerY + 2);
    } while (IsSuperValid(x, y) == false);
    superfood[i] = { x,y };
```

```
  }
}
```

• Hàm `Undying()`: Thao tác khi rắn ăn superfood undying

```
void Undying()
{
  supercount1++;
  if (supercount1 == 40)
  {
    undying = 0;
    supercount1 = 0;
  }
}
```

• Hàm `StopTime()`: Thao tác khi rắn ăn superfood stoptime

```
void StopTime()
{
  supercount++;
  if (supercount == 40)
  {
    stoptime = 0;
    supercount = 0;
  }
}
```

• Hàm `SuperFood()`: Cơ chế ăn superfood và in superfood

```
void SuperFood()
{
  for (int i = 0; i < MAX_SUPER_FOOD; i++)
  {
    int is_eated = 0;
    for (int j = 0; j < eated; j++)
    {
      if (i == superfood_eated[j])
      {
        is_eated = 1;
        break;
      }
    }
    if (is_eated == 1)
    {
      continue;
    }
    if (i == MAX_SUPER_FOOD - 1)
    {
      TextColor(4);
    }
    else
    {
      TextColor(5);
    }
    GotoXY(superfood[i].x, superfood[i].y);
    cout << char(254);
  }
  TextColor(TEXT_COLOR);
  for (int i = 0; i < MAX_SUPER_FOOD; i++)
  {
    int repeated = 0;
    if (snake[SIZE_SNAKE - 1].x == superfood[i].x && snake[SIZE_SNAKE - 1].y
    == superfood[i].y)
```

```
    {
      for (int j = 0; j < eated; j++)
      {
        if (i == superfood_eated[j])
        {
          repeated = 1;
          break;
        }
      }
      if (repeated == 1)
      {
        continue;
      }
      if (i == 4)
      {
        undying = 1;
      }
      else
      {
        stoptime = 1;
        supercount = 0;
      }
      superfood_eated[eated] = i;
      eated++;
      sUperfood1 = 1;
      GotoXY(superfood[i].x, superfood[i].y);
      cout << " ";
    }
  }
}
```

Mảng "superfood" chứa 5 phần tử tương ứng với 4 thức ăn "dừng thời gian" (4 phần tử đầu tiên của mảng) và 1 thức ăn "bất tử" (phần tử cuối cùng của mảng). Hàm "SuperFood" đầu tiên thực hiện vẽ các "siêu thức ăn" vòng lặp lần lượt các phần tử của mảng, trước khi vẽ ta kiểm tra xem phần tử thức ăn đó đã được ăn hay chưa (ta có mảng "superfood_eated" chứa các vị trí trong mảng của các phần tử "superfood"). Nếu rắn ăn "siêu thức ăn", ta cho vị trí trong mảng của phần tử đó vào mảng "superfood_eated", thực hiện gán giá trị tương ứng với chức năng của "siêu thức ăn" đó và cuối cùng, ta thực hiện xóa các "siêu thức ăn" đã được ăn.

2.8.2 Nhóm hàm xử lý vật thể (object)

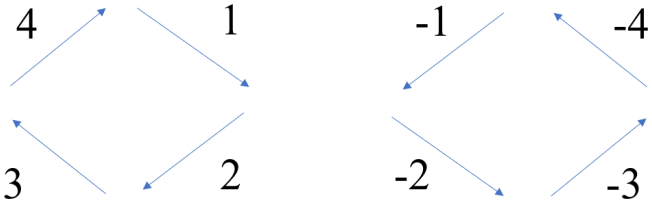
Vật thể Level 3

```
void Object()
{
  int n = 4;
  if (FOOD_INDEX >= 2 && FOOD_INDEX <= 3)
  {
    n = 2;
  }
  if (FOOD_INDEX == 4)
  {
    n = 1;
  }
  for (int i = 0; i < n; i++)
  {
    GotoXY(object[i].x, object[i].y);
    cout << " ";
```

```
  }
  if (object[0].y - 1 == cornerY)
  {
    if (object_state == 4 || object_state == -3)
    {
      object_state = 1;
    }
    else
    {
      object_state = -1;
    }
  }
  if (object[0].x + 2 == cornerX + WIDTH_CONSOLE)
  {
    if (object_state == 1 || object_state == -2)
    {
      object_state = 2;
    }
    else
    {
      object_state = -4;
    }
  }
  if (object[0].y + 2 == cornerY + HEIGH_CONSOLE)
  {
    if (object_state == 2 || object_state == -1)
    {
      object_state = 3;
    }
    else
    {
      object_state = -3;
    }
  }
  if (object[0].x - 1 == cornerX)
  {
    if (object_state == 3 || object_state == -4)
    {
      object_state = 4;
    }
    else
    {
      object_state = -2;
    }
  }
  switch (object_state)
  {
  case 1: case -2:
    for (int i = 0; i < n; i++)
    {
      object[i].x++;
      object[i].y++;
    }
    break;
  case 2: case -1:
    for (int i = 0; i < n; i++)
    {
      object[i].x--;
      object[i].y++;
    }
```

```
76      break;
77    case 3: case -4:
78      for (int i = 0; i < n; i++)
79      {
80        object[i].x--;
81        object[i].y--;
82      }
83      break;
84    case 4: case -3:
85      for (int i = 0; i < n; i++)
86      {
87        object[i].x++;
88        object[i].y--;
89      }
90      break;
91    default:
92      break;
93    }
94    TextColor(11);
95    for (int i = 0; i < n; i++)
96    {
97      GotoXY(object[i].x, object[i].y);
98      cout << (char)219;
99    }
100   TextColor(TEXT_COLOR);
101   if (FOOD_INDEX >= 2 && FOOD_INDEX <= 3)
102   {
103     if (object_fix == 0)
104     {
105       object_state2 = -object_state;
106       object_fix = 1;
107     }
108     DoubleObject();
109   }
110   if (FOOD_INDEX == 4)
111   {
112     if (object_fix1 == 0)
113     {
114       object_state3 = -object_state2;
115       object_state1 = -object_state;
116       object_fix1 = 1;
117     }
118     SingleObject(object_state3, 3);
119     SingleObject(object_state2, 2);
120     SingleObject(object_state1, 1);
121   }
122 }
```

Đầu tiên, "object" là một mảng gồm 4 phần tử liền kề nhau tạo thành hình vuông, các phần tử được khởi tạo lần lượt từ trái qua phải và từ trên xuống. Trạng thái di chuyển của từng phần tử sẽ được tượng trưng lần lượt bằng các biến "object_state", "object_state1", "object_state2" và :object_state3". Các biến này sẽ nhận các giá trị tương ứng với trạng thái di chuyển như hình dưới đây:



Hình 2.1: Các trạng thái di chuyển của object

Ý tưởng cho vật thể ở level 3 là khi rắn ăn 2/4 thức ăn, vật thể sẽ tách làm đôi và khi rắn ăn hết thức ăn, vật thể sẽ tách làm 4. Ta sẽ dùng "object[0]" làm gốc, ta khai báo biến n nhận số phần tử của vật thể là 4, khi FOOD_INDEX = 2 ta gán n = 2 và khi FOOD_INDEX = 4, n sẽ bằng 1. Ở đây ta thực hiện thao tác giống như khi vẽ rắn, đó là xóa vật thể ở vị trí cũ và vẽ vật thể ở vị trí mới trong mỗi vòng lặp. Ở giữa 2 bước này ta gán các vị trí mới cho các phần tử của "object" bằng cách sử dụng "switch", như hình trên ta thấy có 4 hướng di chuyển chính của vật thể tương ứng với 4 cách gán giá trị cho các phần tử (vật thể di chuyển chéo). Tuy vật thể chỉ có 4 hướng di chuyển nhưng các biến tượng trưng cho trạng thái di chuyển của vật thể lại bao gồm 8 giá trị là vì đối với mỗi bức tường trong 4 bức tường sẽ có 2 hướng mà vật thể đập vào tương ứng với 2 hướng bật ra. Vì vậy, trước khi sử dụng "switch" ta cần xét xem vật thể có đang chạm vào tường hay không, nếu có thì ta tiếp tục xét xem trạng thái di chuyển của vật thể hiện tại (biến "object_state") thuộc trường hợp nào trong 2 trường hợp đập vào tường, sau đó thực hiện cập nhật trạng thái di chuyển mới tương ứng với hướng bật ra.

Khi n = 2 (tức là vật thể đã chia làm đôi), đến bước hiện tại ta chỉ mới xóa các phần tử cũ và vẽ các phần tử mới cho vật thể đầu tiên (gồm "object[0]" và "object[1]", ta gọi hàm "DoubleObject" để thực hiện thao tác trên đối với vật thể còn lại.

```
1  void DoubleObject()
2  {
3    for (int i = 2; i < 4; i++)
4    {
5      GotoXY(object[i].x, object[i].y);
6      cout << " ";
7    }
8    if (object[2].y - 1 == cornerY)
9    {
10     if (object_state2 == 4 || object_state2 == -3)
11     {
12       object_state2 = 1;
13     }
14     else
15     {
16       object_state2 = -1;
17     }
18   }
19   if (object[2].x + 2 == cornerX + WIDTH_CONSOLE)
20   {
21     if (object_state2 == 1 || object_state2 == -2)
22     {
23       object_state2 = 2;
24     }
25     else
26     {
```

```
27        object_state2 = -4;
28      }
29    }
30    if (object[2].y + 1 == cornerY + HEIGH_CONSOLE)
31    {
32      if (object_state2 == 2 || object_state2 == -1)
33      {
34        object_state2 = 3;
35      }
36      else
37      {
38        object_state2 = -3;
39      }
40    }
41    if (object[2].x - 1 == cornerX)
42    {
43      if (object_state2 == 3 || object_state2 == -4)
44      {
45        object_state2 = 4;
46      }
47      else
48      {
49        object_state2 = -2;
50      }
51    }
52    switch (object_state2)
53    {
54    case 1: case -2:
55      for (int i = 2; i < 4; i++)
56      {
57        object[i].x++;
58        object[i].y++;
59      }
60      break;
61    case 2: case -1:
62      for (int i = 2; i < 4; i++)
63      {
64        object[i].x--;
65        object[i].y++;
66      }
67      break;
68    case 3: case -4:
69      for (int i = 2; i < 4; i++)
70      {
71        object[i].x--;
72        object[i].y--;
73      }
74      break;
75    case 4: case -3:
76      for (int i = 2; i < 4; i++)
77      {
78        object[i].x++;
79        object[i].y--;
80      }
81      break;
82    default:
83      break;
84    }
85    TextColor(11);
86    for (int i = 2; i < 4; i++)
```

```
87    {
88      GotoXY(object[i].x, object[i].y);
89      cout << (char)219;
90    }
91  }
```

Các thao tác trong hàm "DoubleObject" hoàn toàn tương tự với hàm "Object" tạo nên chuyển động cho vật thể thứ 2 (gồm "object[2]" và "object[3]"). Trong hàm này ta sẽ lấy "object[2]" làm gốc, để quyết định trạng thái di chuyển của vật thể này khi tách ra, ta xét biến cờ object_fix (được khởi tạo bằng 0) nếu bằng 0 sẽ thực hiện gán "object_state2 = -object_state" để vật thể thứ 2 luôn tách ra theo hướng tạo với vật thể thứ nhất một góc 90 độ, sau đó ta gán biến cờ "object_fix = 1" vì đã hoàn thành mục đích.

Khi n = 1 (vật thể chia làm 4), ta thực hiện gọi 3 hàm "SingleObject" để tạo nên chuyển động cho 3 vật thể còn lại.

```
1  void SingleObject(int& n, int i)
2  {
3    GotoXY(object[i].x, object[i].y);
4    cout << " ";
5    if (object[i].y - 1 == cornerY)
6    {
7      if (n == 4 || n == -3)
8      {
9        n = 1;
10     }
11     else
12     {
13       n = -1;
14     }
15   }
16   if (object[i].x + 1 == cornerX + WIDTH_CONSOLE)
17   {
18     if (n == 1 || n == -2)
19     {
20       n = 2;
21     }
22     else
23     {
24       n = -4;
25     }
26   }
27   if (object[i].y + 1 == cornerY + HEIGH_CONSOLE)
28   {
29     if (n == 2 || n == -1)
30     {
31       n = 3;
32     }
33     else
34     {
35       n = -3;
36     }
37   }
38   if (object[i].x - 1 == cornerX)
39   {
40     if (n == 3 || n == -4)
41     {
42       n = 4;
43     }
44     else
45     {
```

```
      n = -2;
    }
  }
  switch (n)
  {
  case 1: case -2:
    object[i].x++;
    object[i].y++;
    break;
  case 2: case -1:
    object[i].x--;
    object[i].y++;
    break;
  case 3: case -4:
    object[i].x--;
    object[i].y--;
    break;
  case 4: case -3:
    object[i].x++;
    object[i].y--;
    break;
  default:
    break;
  }
  TextColor(11);
  GotoXY(object[i].x, object[i].y);
  cout << (char)219;
  TextColor(TEXT_COLOR);
}
```

Hàm "SingleObject" nhận 2 tham số là trạng thái di chuyển (tham chiếu) và vị trí trong mảng "object" (tham trị), hoàn toàn giống với cơ chế di chuyển của hàm "Object" hay "DoubleObject". Ta cũng sử dụng biến cờ "object_fix1" để thực hiện gán "object_state1 = -object_state" và "object_state3 = -object_state2", sau đó gán "object_fix1 = 1". Cuối cùng ta gọi lần lượt 3 hàm "SingleObject" với tham số tương ứng cho từng vật thể.

• Hàm `HitObject()`: Kiểm tra rắn có chạm vào vật thể hay không, nếu rắn chạm vào vật thể ta gọi hàm "ProcessDead". Khi "undying" = 1 ta sẽ không kiểm tra, biến "dead' được dùng cho phần âm thanh.

```
void HitObject()
{
  int dead = 0;
  if (!undying)
  {
    for (int i = 0; i < SIZE_SNAKE; i++)
    {
      for (int j = 0; j < 4; j++)
      {
        if (snake[i].x == object[j].x && snake[i].y == object[j].y)
        {
          ProcessDead();
          dead = 1;
          break;
        }
      }
      if (dead == 1)
      {
        break;
      }
```

```
    }
  }
}
```

Vật thể Level 4

Màn chơi này sẽ có một vật thể (tương tự với vật thể ở level 3) là mảng "ball" gồm 2 phần tử liền kề. Tuy nhiên, ta sẽ không né tránh vật thể này mà sẽ cố gắng điều khiển nó đập vào các bức tường nhỏ, phá hủy chúng và cuối cùng đưa "quả bóng" vào "khung thành" phía sau những bức tường để hoàn thành màn chơi.

```
 void Ball()
{
  for (int i = 0; i < 2; i++)
  {
    GotoXY(ball[i].x, ball[i].y);
    cout << " ";
  }
//up
  for (int i = 0; i < SIZE_SNAKE; i++)
  {
    if ((ball[0].y - 1 == snake[i].y && ball[0].x == snake[i].x) || (ball[1
   ].y - 1 == snake[i].y && ball[1].x == snake[i].x))
    {
      if (ball_state == 4 || ball_state == -3)
      {
        ball_state = 1;
      }
      else
      {
        ball_state = -1;
      }
      break;
    }
  }
  for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
  {
    int hitted = 0;
    for (int k = 0; k < smallwall_hitted_idx; k++)
    {
      if (smallwall_hitted[k] == i)
      {
        hitted = 1;
        break;
      }
    }
    if (hitted == 1)
    {
      continue;
    }
    for (int j = 0; j < 10; j++)
    {
      if ((ball[0].y - 1 == smallwall[i].y && ball[0].x == smallwall[i].x +
   j) || (ball[1].y - 1 == smallwall[i].y && ball[1].x == smallwall[i].x + j
   ))
      {
        if (ball_state == 4 || ball_state == -3)
        {
          ball_state = 1;
```

```
        }
        else
        {
          ball_state = -1;
        }
        smallwall_hitted[smallwall_hitted_idx++] = i;
        break;
      }
    }
  }
  for (int i = 0; i < 2; i++)
  {
    if ((ball[i].x == goal[11].x && ball[i].y - 1 == goal[11].y) || (ball[i
     ].x == goal[13].x && ball[i].y - 1 == goal[13].y))
    {
      if (ball_state == 4 || ball_state == -3)
      {
        ball_state = 1;
      }
      else
      {
        ball_state = -1;
      }
      break;
    }
  }
  if (ball[0].y - 1 == cornerY)
  {
    if (ball_state == 4 || ball_state == -3)
    {
      ball_state = 1;
    }
    else
    {
      ball_state = -1;
    }
  }
//right
  for (int i = 0; i < SIZE_SNAKE; i++)
  {
    if (ball[0].x + 2 == snake[i].x && ball[0].y == snake[i].y)
    {
      if (ball_state == 1 || ball_state == -2)
      {
        ball_state = 2;
      }
      else
      {
        ball_state = -4;
      }
      break;
    }
  }
  for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
  {
    int hitted = 0;
    for (int k = 0; k < smallwall_hitted_idx; k++)
    {
      if (smallwall_hitted[k] == i)
      {
```

```
105         hitted = 1;
106         break;
107       }
108     }
109     if (hitted == 1)
110     {
111       continue;
112     }
113     if (ball[0].x + 2 == smallwall[i].x && ball[0].y == smallwall[i].y)
114     {
115       if (ball_state == 1 || ball_state == -2)
116       {
117         ball_state = 2;
118       }
119       else
120       {
121         ball_state = -4;
122       }
123       smallwall_hitted[smallwall_hitted_idx++] = i;
124     }
125   }
126   if (ball[0].x + 2 == cornerX + WIDTH_CONSOLE)
127   {
128     if (ball_state == 1 || ball_state == -2)
129     {
130       ball_state = 2;
131     }
132     else
133     {
134       ball_state = -4;
135     }
136   }
137 //down
138   for (int i = 0; i < SIZE_SNAKE; i++)
139   {
140     if ((ball[0].y + 1 == snake[i].y && ball[0].x == snake[i].x) || (ball[1
     ].y + 1 == snake[i].y && ball[1].x == snake[i].x))
141     {
142       if (ball_state == 2 || ball_state == -1)
143       {
144         ball_state = 3;
145       }
146       else
147       {
148         ball_state = -3;
149       }
150       break;
151     }
152   }
153   for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
154   {
155     int hitted = 0;
156     for (int k = 0; k < smallwall_hitted_idx; k++)
157     {
158       if (smallwall_hitted[k] == i)
159       {
160         hitted = 1;
161         break;
162       }
163     }
```

```
164     if (hitted == 1)
165     {
166       continue;
167     }
168     for (int j = 0; j < 10; j++)
169     {
170       if ((ball[0].y + 1 == smallwall[i].y && ball[0].x == smallwall[i].x +
      j) || (ball[1].y + 1 == smallwall[i].y && ball[1].x == smallwall[i].x + j
      ))
171       {
172         if (ball_state == 2 || ball_state == -1)
173         {
174           ball_state = 3;
175         }
176         else
177         {
178           ball_state = -3;
179         }
180         smallwall_hitted[smallwall_hitted_idx++] = i;
181         break;
182       }
183     }
184   }
185   if (ball[0].y + 1 == cornerY + HEIGH_CONSOLE)
186   {
187     if (ball_state == 2 || ball_state == -1)
188     {
189       ball_state = 3;
190     }
191     else
192     {
193       ball_state = -3;
194     }
195   }
196 //left
197   for (int i = 0; i < SIZE_SNAKE; i++)
198   {
199     if (ball[0].x - 1 == snake[i].x && ball[0].y == snake[i].y)
200     {
201       if (ball_state == 3 || ball_state == -4)
202       {
203         ball_state = 4;
204       }
205       else
206       {
207         ball_state = -2;
208       }
209       break;
210     }
211   }
212   for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
213   {
214     int hitted = 0;
215     for (int k = 0; k < smallwall_hitted_idx; k++)
216     {
217       if (smallwall_hitted[k] == i)
218       {
219         hitted = 1;
220         break;
221       }
```

```
222       }
223       if (hitted == 1)
224       {
225         continue;
226       }
227       if (ball[0].x - 1 == smallwall[i].x && ball[0].y == smallwall[i].y)
228       {
229         if (ball_state == 3 || ball_state == -4)
230         {
231           ball_state = 4;
232         }
233         else
234         {
235           ball_state = -2;
236         }
237         smallwall_hitted[smallwall_hitted_idx++] = i;
238       }
239     }
240     if (ball[0].x - 1 == cornerX)
241     {
242       if (ball_state == 3 || ball_state == -4)
243       {
244         ball_state = 4;
245       }
246       else
247       {
248         ball_state = -2;
249       }
250     }
251     if (ball[1].x + 1 == goal[11].x && ball[1].y == goal[11].y)
252     {
253       ball_state = 2;
254     }
255     if (ball[0].x - 1 == goal[13].x && ball[0].y == goal[13].y)
256     {
257       ball_state = 1;
258     }
259     switch (ball_state)
260     {
261     case 1: case -2:
262       for (int i = 0; i < 2; i++)
263       {
264         ball[i].x++;
265         ball[i].y++;
266       }
267       break;
268     case 2: case - 1:
269       for (int i = 0; i < 2; i++)
270       {
271         ball[i].x--;
272         ball[i].y++;
273       }
274       break;
275     case 3: case -4:
276       for (int i = 0; i < 2; i++)
277       {
278         ball[i].x--;
279         ball[i].y--;
280       }
281       break;
```

```
282   case 4: case -3:
283     for (int i = 0; i < 2; i++)
284     {
285       ball[i].x++;
286       ball[i].y--;
287     }
288     break;
289   default:
290     break;
291   }
292   TextColor(11);
293   for (int i = 0; i < 2; i++)
294   {
295     GotoXY(ball[i].x, ball[i].y);
296     cout << (char)219;
297   }
298   TextColor(TEXT_COLOR);
299 }
300
```

Đầu tiên, ta sẽ có hàm "Ball" để tạo nên cơ chế di chuyển của "quả bóng". Hoàn toàn tương tự với vật thể ở level 3 ta cũng thực hiện theo các bước xóa "quả bóng" ở vị trí cũ, kiểm tra các điều kiện để cập nhật trạng thái di chuyển với biến "ball_state" (ta sử dụng các giá trị tượng trưng cho hướng giống với vật thể ở level 3), tạo vị trí mới tương ứng với trạng thái di chuyển và cuối cùng là vẽ "quả bóng" ở vị trí mới.

Tuy nhiên, ngoài các điều kiện đập vào tường, ta cần xét thêm các điều kiện khi đập vào rắn, các bức tường nhỏ và khung thành.

Đối với trường hợp "quả bóng" đập vào rắn, ta thực hiện các vòng lặp "for" để kiểm tra quả bóng có đập vào rắn với 1 trong 4 hướng hay không, nếu có ta tiếp tục xét trạng thái di chuyển hiện tại của quả bóng ("ball_state") để cập nhật trạng thái di chuyển mới tương ứng trong 2 trường hợp, hoàn toàn tương tự với trường hợp đập vào tường.

Ta tiếp tục xét trường hợp quả bóng đập vào các bức tường nhỏ, mảng "smallwall" gồm có 28 phần tử tương đương với 28 bức tường được xếp thành 4 dòng 7 cột ở phía trên trong khung game. Ta cũng có mảng "smallwall_hitted" chứa các vị trí của các phần tử trong mảng "smallwall" sau khi bị quả bóng đập vào. Ta cần xét 4 trường hợp tương ứng với 4 hướng đập vào của quả bóng, đối với cả 4 trường hợp, ta dùng vòng "for" để truy xuất lần lượt các phần tử của mảng "smallwall", sau đó kiểm tra xem vị trí đó đã có trong mảng "smallwall_hitted" hay chưa, nếu có thì ta dùng lệnh "continue" để tiếp tục truy xuất tới những phần tử tiếp theo. Đối với trường hợp bóng đập lên và đập xuống, ta kiểm tra bóng có chạm vào khoảng từ phần tử "smallwall" đó kéo dài đến 9 kí tự tiếp theo bằng vòng lặp "for" (vì các bức tường được vẽ kéo dài ra 10 kí tự tính từ vị trí của nó), nếu các điều kiện đã thỏa mãn, ta cập nhật trạng thái di chuyển "ball_state" và kết thúc thúc vòng lặp bằng lệnh "break". Đối với trường hợp đập vào cạnh bên của các bức tường, ta không cần dùng vòng lặp "for" vì chỉ có duy nhất một trường hợp cần xét. Cuối cùng, ta thêm vị trí trong mảng "smallwall" của phần tử tường đang xét vào mảng "smallwall_hitted" và tăng biến số phần tử "smallwall_hitted_idx" thêm 1 đơn vị.

Cuối cùng , ta xét đến các trường hợp bóng đập vào khung thành (mảng "goal" gồm 14 phần tử đã được khởi tạo trước đó ở giữa bức tường phía trên). Tương tự như trên, ta cũng cập nhật trạng thái di chuyển "ball_state" mỗi khi điều kiện bóng đập vào khung thành được thỏa mãn.

```
1 void SmallWallHitted()
2 {
3   for (int i = 0; i < smallwall_hitted_idx; i++)
4   {
```

```
    GotoXY(smallwall[smallwall_hitted[i]].x, smallwall[smallwall_hitted[i]].y);
    if (smallwall_hitted[i] == 6 || smallwall_hitted[i] == 13 || smallwall_hitted[i] == 20 || smallwall_hitted[i] == 27)
    {
      cout << "         ";
    }
    else
    {
      cout << "          ";
    }
  }
}

void HitSmallWall()
{
  for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
  {
    int hitted = 0;
    for (int k = 0; k < smallwall_hitted_idx; k++)
    {
      if (i == smallwall_hitted[k])
      {
        hitted = 1;
        break;
      }
    }
    if (hitted == 1)
    {
      continue;
    }
    for (int j = 0; j < 10; j++)
    {
      if (snake[SIZE_SNAKE - 1].x == smallwall[i].x + j && snake[SIZE_SNAKE - 1].y == smallwall[i].y)
      {
        ProcessDead();
        break;
      }
    }
  }
  if ((snake[SIZE_SNAKE - 1].x == goal[11].x && snake[SIZE_SNAKE - 1].y == goal[11].y) || (snake[SIZE_SNAKE - 1].x == goal[13].x && snake[SIZE_SNAKE - 1].y == goal[13].y))
  {
    ProcessDead();
  }
  for (int i = 1; i < 11; i++)
  {
    if (snake[SIZE_SNAKE - 1].x == goal[11].x + i && snake[SIZE_SNAKE - 1].y == goal[11].y)
    {
      ProcessDead();
      break;
    }
  }
}
```

Cùng với "Ball", ta gọi hàm "SmallWallHitted" để xóa những bức tường đã bị bóng đập vào bằng cách truy xuất đến các phần tử của mảng "smallwall_hitted" và vẽ cả khoảng trắng

tương ứng.

Hàm "HitSmallWall" gọi hàm "ProcessDead" mỗi khi rắn chạm vào các bức tường (chưa bị phá) hoặc khung thành, các thao tác hoàn toàn tương tự với các hàm phía trên.

```
void BreakLength()
{
  if (length_break < smallwall_hitted_idx)
  {
    BreakWall();
    length_break++;
  }
}

void BreakWall()
{
    SIZE_SNAKE++;
    SIZE_SNAKE_1++;
    SCORE++;
  switch (MOVING)
  {
  case 'A':
    snake[SIZE_SNAKE] = { snake[SIZE_SNAKE - 1].x - 1, snake[SIZE_SNAKE - 1
   ].y };
    for (int i = 0; i < SIZE_SNAKE - 1; i++) {
      snake[i].x = snake[i + 1].x;
      snake[i].y = snake[i + 1].y;
    }
    snake[SIZE_SNAKE - 1].x--;
    break;
  case 'W':
    snake[SIZE_SNAKE] = { snake[SIZE_SNAKE - 1].x, snake[SIZE_SNAKE - 1].y -
    1 };
    for (int i = 0; i < SIZE_SNAKE - 1; i++) {
      snake[i].x = snake[i + 1].x;
      snake[i].y = snake[i + 1].y;
    }
    snake[SIZE_SNAKE - 1].y--;
    break;
  case 'D':
    snake[SIZE_SNAKE] = { snake[SIZE_SNAKE - 1].x + 1, snake[SIZE_SNAKE - 1
   ].y };
    for (int i = 0; i < SIZE_SNAKE - 1; i++) {
      snake[i].x = snake[i + 1].x;
      snake[i].y = snake[i + 1].y;
    }
    snake[SIZE_SNAKE - 1].x++;
    break;
  case 'S':
    snake[SIZE_SNAKE] = { snake[SIZE_SNAKE - 1].x, snake[SIZE_SNAKE - 1].y +
    1 };
    for (int i = 0; i < SIZE_SNAKE - 1; i++) {
      snake[i].x = snake[i + 1].x;
      snake[i].y = snake[i + 1].y;
    }
    snake[SIZE_SNAKE - 1].y++;
    break;
  default:
    break;
  }
}
```

Hàm "BreakLength" được gọi để kiểm tra có tường nào bị phá hay không, nếu có ta gọi hàm "BreakWall" để cập nhật độ dài, điểm số và vị trí mới cho rắn hoàn toàn tương tự như khi rắn ăn thức ăn. Hàm "hittedcheck" cũng được gọi chung với các hàm trên đây trong "ThreadFunc" (được định nhĩa trong Gameplay.cpp).

```
void GoalGoalGoal()
{
  if (goalgoalgoal == 0)
  {
    for (int i = 0; i < 11; i++)
    {
      if (ball[0].x == goal[i].x && ball[0].y - 1 == goal[i].y)
      {
        goalSound = 1;
        GotoXY(ball[0].x, ball[0].y);
        cout << "  ";
        goalgoalgoal = 1;
        for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
        {
          GotoXY(smallwall[i].x, smallwall[i].y);
          if (i == 6 || i == 13 || i == 20 || i == 27)
          {
            cout << "        ";
          }
          else
          {
            cout << "         ";
          }
        }
        break;
      }
    }
  }
  if (goal_celebration < 15 && goalgoalgoal == 1)
  {
    if (goal_celebration % 2 == 0)
    {
      TextColor(4);
      DrawGoal();
      TextColor(TEXT_COLOR);
      goal_celebration++;
    }
    else
    {
      DrawGoal();
      goal_celebration++;
    }
  }
  if (goal_celebration == 15 && goalgoalgoal == 1)
  {
    TextColor(3);
    for (int i = 0; i < 10; i++)
    {
      GotoXY(goal[i].x, goal[i].y);
      cout << char(205);
    }
    GotoXY(goal[10].x, goal[10].y); cout << char(205);
    GotoXY(goal[11].x, goal[11].y); cout << "            ";
    GotoXY(goal[12].x, goal[12].y); cout << char(205);
    TextColor(TEXT_COLOR);
```

```
    GATE = 1;
    goal_celebration++;
  }
}
```

Hàm "GoalGoalGoal" liên tục kiểm tra bóng đã đi vào khung thành hay chưa, nếu điều kiện thỏa mãn, ta thực hiện gán biến "goalgoalgoal = 1" để dừng việc gọi các hàm phía trên cũng như dừng việc kiểm tra bóng đã vào khung thành hay chưa, xóa bóng và các bức tường chưa bị phá, biến "goalSound" cũng được gán bằng 1 để phục vụ cho phần âm thanh. Sau đó ta tạo hiệu ứng cho khung thành nhấp nháy trước khi biến mất và gán "GATE = 1" để mở cổng và hoàn thành màn chơi.

Cuối cùng ta có hàm `DelWal`: vá lỗi hiển thị tường khi chơi tiếp level 4

```
void DelWall() {
  for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++)
  {
    GotoXY(smallwall[i].x, smallwall[i].y);
    if (i == 6 || i == 13 || i == 20 || i == 27)
    {
      cout << "         ";
    }
    else
    {
      cout << "          ";
    }
  }
  TextColor(3);
  for (int i = 0; i < 10; i++)
  {
    GotoXY(goal[i].x, goal[i].y);
    cout << char(205);
  }
  GotoXY(goal[10].x, goal[10].y); cout << char(205);
  GotoXY(goal[11].x, goal[11].y); cout << "              ";
  GotoXY(goal[12].x, goal[12].y); cout << char(205);
  TextColor(TEXT_COLOR);
}
```

2.9 Nhóm hàm khai báo trong file Gameplay.h

Ở mục này, mã nguồn của các hàm sẽ được trình bày cụ thể theo từng Level.

2.9.1 Level 1

- **ResetData**

```
void ResetData()
{
  CHAR_LOCK = 'A', MOVING = 'D', SPEED = 1; FOOD_INDEX = 0, WIDTH_CONSOLE =
   71, HEIGH_CONSOLE = 20, SIZE_SNAKE = 0;
  GATE = 0;
  SCORE = 0;
  SIZE_SNAKE_1 = 6;
  GenerateFood();
  GateInit();
  out[0] = { cornerX, 10 + cornerY };
  flag = 0;
```

```
  outgate = 0;
  colorboard = 0;
  level = 1;
  fix = 1;
  fixcount = 0;
  SPEED = play_again;
}

void GateInit()
{
  int x, y;
  srand(time(NULL));
  x = rand() % 2;
  if (x == 0)
  {
    x = rand() % (WIDTH_CONSOLE - 1) + 1 + cornerX;
    y = rand() % 2;
    if (y == 0) // top
    {
      y = cornerY;
      gate[0] = { x, y };
      gate[1] = { x + 1, y };
      gate[2] = { x - 1, y };
      gate[3] = { x + 1, y + 1 };
      gate[4] = { x - 1, y + 1 };
    }
    else // bottom
    {
      y = HEIGH_CONSOLE + cornerY;
      gate[0] = { x, y };
      gate[1] = { x + 1, y };
      gate[2] = { x - 1, y };
      gate[3] = { x + 1, y - 1 };
      gate[4] = { x - 1, y - 1 };
    }
  }
  else
  {
    y = rand() % (HEIGH_CONSOLE - 1) + 1 + cornerY;
    x = rand() % 2;
    if (x == 0) // left
    {
      x = cornerX;
      gate[0] = { x, y };
      gate[1] = { x, y + 1 };
      gate[2] = { x, y - 1 };
      gate[3] = { x + 1, y + 1 };
      gate[4] = { x + 1, y - 1 };
    }
    else // right
    {
      x = WIDTH_CONSOLE + cornerX;
      gate[0] = { x, y };
      gate[1] = { x, y + 1 };
      gate[2] = { x, y - 1 };
      gate[3] = { x - 1, y + 1 };
      gate[4] = { x - 1, y - 1 };
    }
  }
}
```

Hàm "ResetData" dùng để reset cũng như khởi tạo các biến cần thiết trước khi bắt đầu level 1. Hàm này sẽ gọi hàm "GenerateFood" dùng để khởi tạo vị trí của thức ăn và "GateInit" để khởi tạo cổng để con rắn chui vào khi ăn đủ thức ăn.

Đối với hàm "GateInit", ta khởi tạo hai biến kiểu int x và y đồng thời gọi hàm "srand(time(NULL))", sau đó ta cho x nhận giá trị ngẫu nhiên 0 hoặc 1, nếu x bằng 0 thì cổng sẽ được khởi tạo ở tường phía trên hoặc tường phía dưới, sau đó ta cho x nhận giá trị ngẫu nhiên trên cạnh tường (từ cornerX + 1 đến cornerX + WIDTH_CONSOLE - 1), lúc này ta cho y nhận giá trị ngẫu nhiên 0 hoặc 1 để quyết định cổng sẽ nằm ở tường phía trên hay tường phía dưới, nếu y bằng 0 thì cổng sẽ nằm ở phía trên, ta gán y = cornerY còn nếu y bằng 1 thì cổng nằm ở phía dưới, ta gán y = HEIGH_CONSOLE + cornerY. Sau đó ta lần lượt gán giá trị cho các phần tử của mảng "gate" (gồm 5 phần tử) sao cho các phần tử tạo thành hình dạng của cổng. Đối với trường hợp x = 1 ta sẽ thực hiện các bước tương tự để khởi tạo cổng ở cạnh bên trái hoặc bên phải.

- **StartGame**

```
void StartGame() {
  system("cls");
  ResetData(); // Intialize original data
  DrawFrame(corFx, corFy);
  DrawLeftBoard(corLx, corLy);
  DeleteFrameLevel(corLevelx, corLevely);
  DrawFrameLevel(corLevelx, corLevely);
  TextColor(3);
  DrawBoard(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE); // Draw game
  STATE = 1;//Start running Thread
}
```

Tiếp theo, hàm "StartGame" sẽ thực hiện những thao tác cần thiết trước khi bắt đầu level 1. Dòng đầu tiên sẽ xóa màn hình cũ, sau đó ta gọi hàm "ResetData" để khởi tạo các dữ liệu cần thiết. Tiếp theo đó là các hàm "DrawFrame", "DrawLeftBoard", "DeleteFrameLevel", "DrawFrameLevel" đã được nói đến ở phần nhóm hàm khai báo trong file Draw.h. Hàm "TextColor(3)" và "DrawBoard" dùng để vẽ màn chơi cho level 1 (khung bao quanh khu vực rắn có thể di chuyển). Và cuối cùng là một dòng rất quan trọng, gán biến STATE = 1, rắn xuất hiện và trò chơi bắt đầu.

2.9.2 Level 2

- **ResetData1**

```
void ResetData1()
{
  CHAR_LOCK = 'A', MOVING = 'D', SPEED = 1; FOOD_INDEX = 0, WIDTH_CONSOLE =
   71, HEIGH_CONSOLE = 20, SIZE_SNAKE = 0;
  GATE = 0;
  WallInit();
  GenerateFood1();
  GateInit();
  out[0] = { cornerX,10 + cornerY };
  flag = 0;
  outgate = 0;
  fix = 1;
  colorboard = 0;
  level = 2;
  fixcount = 0;
  SPEED = play_again;
}
```

```
17
18 void WallInit()
19 {
20   int n = 1;
21   wall[0] = { WIDTH_CONSOLE / 2 + 28, HEIGH_CONSOLE / 2 + 3 };
22   wall[n++] = { wall[0].x + 5, wall[0].y };
23   wall[n++] = { wall[0].x, wall[0].y + 5 };
24   wall[n++] = { wall[0].x + 5, wall[0].y + 5 };
25   for (int i = 1; i < 5; i++)
26   {
27     //top left
28     wall[n++] = { wall[0].x - i, wall[0].y };
29     wall[n++] = { wall[0].x, wall[0].y - i };
30     //top right
31     wall[n++] = { wall[0].x + 5 + i, wall[0].y };
32     wall[n++] = { wall[0].x + 5, wall[0].y - i };
33     //bottom left
34     wall[n++] = { wall[0].x - i, wall[0].y + 5 };
35     wall[n++] = { wall[0].x, wall[0].y + 5 + i };
36     //bottom right
37     wall[n++] = { wall[0].x + 5 + i, wall[0].y + 5 };
38     wall[n++] = { wall[0].x + 5, wall[0].y + 5 + i };
39   }
40   for (int i = 0; i < 4; i++)
41   {
42     wall[n++] = { wall[0].x - 5 - i, wall[0].y };
43     wall[n++] = { wall[0].x + 10 + i, wall[0].y };
44     wall[n++] = { wall[0].x - 5 - i, wall[0].y + 5 };
45     wall[n++] = { wall[0].x + 10 + i, wall[0].y + 5 };
46   }
47 }
```

Hàm "ResetData1" dùng để reset cũng như khởi tạo các biến cần thiết trước khi bắt đầu level 2. Hàm này sẽ gọi hàm "WallInit" để khởi tạo giá trị cho các phần tử của mảng "wall" (gồm 52 phần tử) dùng cho tường trong level 2. Tiếp theo là hàm "GenerateFood1" để khởi tạo thức ăn cho level 2. Ta cũng gọi hàm "GateInit" tương tự như trong "ResetData".

- **StartGame1**

```
1  void StartGame1() {
2    system("cls");
3    ResetData1(); // Intialize original data
4    DeleteFrameLevel(corLevelx, corLevely);
5    DrawFrame(corFx, corFy);
6    DeleteFrameLevel(corLevelx, corLevely);
7    DrawFrameLevel1(corLevelx, corLevely);
8    DrawLeftBoard(corLx, corLy);
9    TextColor(3);
10   DrawBoard1(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE); // Draw game
11   STATE = 1;//Start running Thread
12 }
```

Hàm "StartGame1" cũng sẽ thực hiện các thao tác trước khi bắt đầu level 2, hoàn toàn tương tự với hàm "StartGame" ngoại trừ việc gọi hàm "ResetData1", "DrawFrameLeve1" và "DrawBoard1".

2.9.3 Level 3

- **ResetData2**

```
void ResetData2()
{
  CHAR_LOCK = 'A', MOVING = 'D', SPEED = 1; FOOD_INDEX = 0, WIDTH_CONSOLE =
   71, HEIGH_CONSOLE = 20, SIZE_SNAKE = 0;
  GATE = 0;
  GenerateFood();
  GateInit();
  out[0] = { cornerX,10 + cornerY };
  flag = 0;
  outgate = 0;
  fix = 1;
  colorboard = 0;
  level = 3;
  fixcount = 0;
  ObjectInit();
  object_state = 4;
  ResetWall();
  object_state1 = 4;
  object_state2 = 1;
  object_state3 = 1;
  GenerateSuperFood();
  SPEED = play_again;
}

void ResetWall()
{
  for (int i = 0; i < SIZE_WALL; i++)
  {
    wall[i] = { i,0 };
  }
}

void ObjectInit()
{
  object[0] = { cornerX + WIDTH_CONSOLE / 2, cornerY + 1 };
  object[1] = { cornerX + WIDTH_CONSOLE / 2 + 1, cornerY + 1 };
  object[2] = { cornerX + WIDTH_CONSOLE / 2, cornerY + 2 };
  object[3] = { cornerX + WIDTH_CONSOLE / 2 + 1, cornerY + 2 };
}
```

Hàm "ResetData2" dùng để reset cũng như khởi tạo các biến cần thiết trước khi bắt đầu level 3. Hàm này cũng gọi các hàm "GenerateFood" và "GateInit" như trong "StartGame". Ở level này, ta sẽ có một vật thể chuyển động liên tục và để khởi tạo nó ta gọi hàm "ObjectInit" gán cho từng phần tử của mảng "object" (gồm 4 phần tử) các vị trí cạnh nhau tạo thành một hình vuông. Ngoài ra, ta sẽ có thêm các "siêu thức ăn" với những tác dụng đặc biệt được khởi tạo từ hàm "GenerateSuperFood" cũng tương tự như "GenerateFood" nhưng được kiểm tra với hàm "IsSuperValid" để không bị trùng với thức ăn bình thường. Ngoài ra, ta cũng cần gọi hàm "ResetWall" để chuyển các phần tử thuộc mảng "wall" ra khỏi khu vực di chuyển của rắn.

- **StartGame2**

```
void StartGame2() {
  system("cls");
  ResetData2(); // Intialize original data  DrawFrame(corFx, corFy);
  DrawFrame(corFx, corFy);
  DeleteFrameLevel(corLevelx, corLevely);
  DrawFrameLevel2(corLevelx, corLevely);
  DrawLeftBoard(corLx, corLy);
```

```
  TextColor(3);
  DrawBoard(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE); // Draw game
  STATE = 1;//Start running Thread
}
```

Hàm "StartGame2" cũng sẽ thực hiện các thao tác trước khi bắt đầu level 3, hoàn toàn tương tự với hàm "StartGame" ngoại trừ việc gọi hàm "ResetData2" và "DrawFrameLeve2".

2.9.4 Level 4

- **ResetData3**

```
void ResetData3()
{
  CHAR_LOCK = 'A', MOVING = 'D', SPEED = 1; FOOD_INDEX = 4, WIDTH_CONSOLE =
   71, HEIGH_CONSOLE = 20, SIZE_SNAKE = 0;
  GATE = 0;
  BallInit();
  SmallWallInit();
  GoalInit();
  ResetObject();
  ResetFood();
  GateInit();
  out[0] = { cornerX,10 + cornerY };
  flag = 0;
  outgate = 0;
  fix = 1;
  colorboard = 0;
  level = 4;
  fixcount = 0;
  SPEED = play_again;
  ball_state = 1;
}

void SmallWallInit()
{
  int idx = 0;
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 7; j++)
    {
      smallwall[idx++] = { cornerX + 2 + 10 * j, cornerY + 2 + i };
    }
  }
}

void GoalInit()
{
  for (int i = 0; i < 10; i++)
  {
    goal[i] = { cornerX + 30 + i, cornerY };
  }
  goal[10] = { goal[0].x - 1, goal[0].y };
  goal[11] = { goal[0].x - 1, goal[0].y + 1 };
  goal[12] = { goal[9].x + 1, goal[9].y };
  goal[13] = { goal[9].x + 1, goal[9].y + 1 };
}

void BallInit()
{
```

```
48    ball[0] = { cornerX + 40, cornerY + 10 };
49    ball[1] = { cornerX + 41, cornerY + 10 };
50  }
51
52  void ResetObject()
53  {
54    for (int i = 0; i < 4; i++)
55    {
56      object[i] = { 1 + i, 0 };
57    }
58  }
59
60  void ResetFood()
61  {
62    for (int i = 0; i < MAX_SIZE_FOOD; i++)
63    {
64      food[i] = { 1 + i, 0 };
65    }
66  }
```

Hàm "ResetData3" dùng để reset cũng như khởi tạo các biến cần thiết trước khi bắt đầu level 4. Ở level này, rắn sẽ không ăn thức ăn nữa mà sẽ đưa quả bóng là mảng "ball" (gồm 2 phần tử liền kề nhau di chuyển liên tục tương tự object ở level 3) đập vào và phá hủy các bức tường nhỏ là mảng "smallwall" (gồm 28 phần tử), và cuối cùng là đưa bóng vào khung thành là mảng "goal" (gồm 14 phần tử) để có thể hoàn thành level. Các hàm được gọi để khởi tạo các mảng nêu trên lần lượt là các hàm "BallInit", "SmallWallInit" và "GoalInit". Ngoài ra, hàm "ResetObject" được gọi để đưa "object" từ level 3 ra khỏi khu vực di chuyển và hàm "ResetFood" cũng được gọi để đưa thức ăn được khởi tạo từ level 3 ra khỏi khu vực di chuyển. Tất nhiên, hàm "GateInit" cũng được gọi để tạo cổng khi hoàn thành level.

- **StartGame3**

```
1  void StartGame3() {
2    system("cls");
3    ResetData3(); // Intialize original data
4    DeleteFrameLevel(corLevelx, corLevely);
5    DrawFrame(corFx, corFy);
6    DeleteFrameLevel(corLevelx, corLevely);
7    DrawFrameLevel3(corLevelx, corLevely);
8    DrawLeftBoard(corLx, corLy);
9    TextColor(3);
10   DrawBoard3(cornerX, cornerY, WIDTH_CONSOLE, HEIGH_CONSOLE); // Draw game
11   STATE = 1;
12 }
```

Hàm "StartGame3" cũng sẽ thực hiện các thao tác trước khi bắt đầu level 4, ngoài các hàm tương tự như "ResetData", hàm "DrawBoard3" sẽ gọi các hàm "DrawSmallWall" để hiển thị các bức tường nhỏ được xếp thành 4 dòng và 7 cột và hàm "DrawGoal" để hiển thị khung thành.

2.9.5 Các hàm khác

- Hàm `DrawGate()`: Vẽ cổng

```
1  void DrawGate()
2  {
3    if (gate[0].x == cornerX)
4    {
5      TextColor(3);
```

```
    GotoXY(gate[0].x, gate[0].y);
    cout << "0";
    TextColor(4);
    GotoXY(gate[1].x, gate[1].y);
    cout << char(200);
    GotoXY(gate[2].x, gate[2].y);
    cout << char(201);
    GotoXY(gate[3].x, gate[3].y);
    cout << char(185);
    GotoXY(gate[4].x, gate[4].y);
    cout << char(185);
    TextColor(TEXT_COLOR);
  }
  else if (gate[0].x == WIDTH_CONSOLE + cornerX)
  {
    TextColor(3);
    GotoXY(gate[0].x, gate[0].y);
    cout << "0";
    TextColor(4);
    GotoXY(gate[1].x, gate[1].y);
    cout << char(188);
    GotoXY(gate[2].x, gate[2].y);
    cout << char(187);
    GotoXY(gate[3].x, gate[3].y);
    cout << char(204);
    GotoXY(gate[4].x, gate[4].y);
    cout << char(204);
    TextColor(TEXT_COLOR);
  }
  else if (gate[0].y == cornerY)
  {
    TextColor(3);
    GotoXY(gate[0].x, gate[0].y);
    cout << "0";
    TextColor(4);
    GotoXY(gate[1].x, gate[1].y);
    cout << char(187);
    GotoXY(gate[2].x, gate[2].y);
    cout << char(201);
    GotoXY(gate[3].x, gate[3].y);
    cout << char(202);
    GotoXY(gate[4].x, gate[4].y);
    cout << char(202);
    TextColor(TEXT_COLOR);
  }
  else
  {
    TextColor(3);
    GotoXY(gate[0].x, gate[0].y);
    cout << "0";
    TextColor(4);
    GotoXY(gate[0].x, gate[0].y);
    GotoXY(gate[1].x, gate[1].y);
    cout << char(188);
    GotoXY(gate[2].x, gate[2].y);
    cout << char(200);
    GotoXY(gate[3].x, gate[3].y);
    cout << char(203);
    GotoXY(gate[4].x, gate[4].y);
    cout << char(203);
```

```
    TextColor(TEXT_COLOR);
  }
}
```

Hàm "DrawGate" thực hiện vẽ cổng từ các phần tử của mảng "gate" đã được khởi tạo ngẫu nhiên từ các hàm "ResetData"

• Ham `GetOut()`: Khởi tạo rắn khi bắt đầu Level tiếp theo

```

void GetOut()
{
  while (SIZE_SNAKE < SIZE_SNAKE_1)
  {
    snake[SIZE_SNAKE] = out[0];
    SIZE_SNAKE++;
  }
}
```

Hàm này thực hiện khởi tạo lại rắn ở đầu ra "out" (nằm ở vị trí cố định được khởi tạo sẵn) bằng một vòng lặp "while (SIZE_SNAKE < SIZE_SNAKE_1)" khởi tạo lần lượt cho từng phần tử của "snake" (SIZE_SNAKE là độ dài theo thời gian thực của rắn giảm về 0 mỗi khi qua cổng nên ta có SIZE_SNAKE_1 là độ dài của rắn được tích lũy trong khi chơi, dùng để phục hồi lại độ dài của rắn mỗi khi bắt đầu một level).

• Các hàm `GateOut()` và `Fix()`:

```
void GateOut()
{
  if (outgate < SIZE_SNAKE_1)
  {
    TextColor(5);
    GotoXY(out[0].x, out[0].y + 1); cout << char(175);
    GotoXY(out[0].x, out[0].y - 1); cout << char(175);
    TextColor(3);
    GotoXY(out[0].x, out[0].y); cout << '0';
    outgate++;
    TextColor(TEXT_COLOR);
    if (outgate == SIZE_SNAKE_1)
    {
      TextColor(3);
      GotoXY(out[0].x, out[0].y + 1); cout << (char)186;
      GotoXY(out[0].x, out[0].y - 1); cout << (char)186;
      GotoXY(out[0].x, out[0].y); cout << (char)186;
      TextColor(TEXT_COLOR);
      if (fix == 1)
      {
        fix = 0;
      }
    }
  }
}

void Fix()
{
  TextColor(3);
  GotoXY(out[0].x, out[0].y); cout << (char)186;
  TextColor(TEXT_COLOR);
}
```

"GateOut" để tạo cổng ra khi bắt đầu màn chơi. Trong hàm này, ta xét điều kiện khi biến outgate < SIZE_SNAKE_1 (gateout được khởi tạo bằng 0), nếu điều kiện thỏa mãn, ta sẽ vẽ cổng ra cho rắn tại vị trí của "out" đồng thời tăng dần giá trị cho biến outgate. Khi outgate = SIZE_SNAKE_1, ta sẽ xóa cổng bằng cách vẽ đè tường lên cổng và gán fix = 0 nếu fix = 1. Các biến fix, fixcount và hàm "Fix" được dùng để vá lại chỗ trống trên tường mà rắn để lại (thao tác này chưa thực sự tối ưu).

• Hàm `PauseGame`: Tạm dừng trò chơi

```
void PauseGame(HANDLE t)
{
  SuspendThread(t);
  TextColor(3);
  ClearBoard();
  TextColor(3);
  if (GATE == 1)
  {
    for (int i = 0; i < 3; i++)
    {
      GotoXY(gate[i].x, gate[i].y);
      if (gate[0].x == cornerX || gate[0].x == cornerX + WIDTH_CONSOLE)
      {
        cout << char(186);
      }
      else
      {
        cout << char(205);
      }
    }
  }
  TextColor(TEXT_COLOR);
  DrawPause();
  int temp;
  while (true)
  {
    temp = toupper(_getch());
    if (temp == 'P')
    {
      ClearPause();
      ResumeThread(t);
      break;
    }
    else if (temp == 'S')
    {
      STATE = 0;
      ClearBoard();
      Result();
      int temp2;
      while (1) {
        temp2 = toupper(_getch());
        if (temp2 == 'S') {
          save_load = 1;
          Save();
          SaveBoard();
          SaveInf(name);
          break;
        }
        else if (temp2 == 'E') {
          save_load = 1;
          break;
```

```
        }
      }
      break;
    }
    else
    {
      continue;
    }
  }
}
```

Hàm này giúp người chơi có thể tạm dừng trò chơi bằng cách nhập phím P từ bàn phím. Cùng với đó là các chức năng sau khi tạm dừng như lưu game hoặc tiếp tục màn chơi. Các hàm được khai báo trong file Draw.h được gọi để thiết kế những giao diên tương ứng với từng chức năng.

- Hàm ThreadFunc:

```
void ThreadFunc() {
  while (1) {
    if (STATE == 1) {//If my snake is alive
      if (flag == 0)
      {
        GetOut();
        flag = 1;
      }
      char ch[] = " ";
      DrawSnake(ch);
      switch (MOVING) {
      case 'A':
        MoveLeft();
        Process();
        break;
      case 'D':
        MoveRight();
        Process();
        break;
      case 'W':
        MoveUp();
        Process();
        break;
      case 'S':
        MoveDown();
        Process();
        break;
      }
      DrawInfo(corLx, corLy + 1);
      if (level == 3)
      {
        HitObject();
        SuperFood();
        if (undying)
        {
          Undying();
        }
        if (stoptime)
        {
          StopTime();
        }
        else
```

```
43          {
44            Object();
45          }
46        }
47        if (level == 4)
48        {
49          if (goalgoalgoal == 0)
50          {
51            HitSmallWall();
52            SmallWallHitted();
53            Ball();
54            hittedcheck();
55            BreakLength();
56          }
57          GoalGoalGoal();
58        }
59        DrawSnakeAndFood(mssv);
60        GateOut();
61        if (fix == 0 && fixcount < SIZE_SNAKE_1 + 2)
62        {
63          Fix();
64          fixcount++;
65        }
66        Sleep(100 / SPEED);
67        if (STATE == 0) {
68          ClearBoard();
69          Result();
70        }
71      }
72    }
73  }
```

Trong hàm này, ta có một vòng lặp thực hiện tất cả thao tác cần thiết của game mỗi khi bắt đầu một level.Trước hết, ta sẽ nói về các thao tác được thực hiện ở tất cả các level, sau đó ta sẽ xét riêng level 3 và level 4. Khi STATE = 1 (tức là rắn còn sống) ta sẽ luôn thực hiện các thao tác sau: (dòng 4 đến dòng 29) và (dòng 59 đến dòng 69).

Đầu tiên, ta xét biến "flag" (luôn được khởi tạo bằng 0 trong các hàm "ResetData"), nếu flag = 0 ta gọi hàm "GetOut".

Tiếp theo, như đã nhắc đến ở trên, ta gọi hàm "DrawSnake" để xóa con rắn ở vị trí cũ trước khi vẽ lại ở vị trí mới. Tiếp đến, ta dùng "switch" để kiểm tra hướng di chuyển của rắn để gọi hàm "Move" tương ứng. Sau đó ta gọi hàm "Process" để thực hiện các thao tác với quá trình qua màn. Kế tiếp ta gọi hàm "GateOut" để tạo cổng ra khi bắt đầu màn chơi.

Sau các thao tác trên, ta cho thread "Sleep(100 / SPEED)", SPEED càng cao thì thread "Sleep" càng ít và trò chơi càng nhanh. Cuối cùng, khi rắn chết (STATE = 0), ta vẽ bảng kết quả bằng cách gọi hàm "ClearBoard" và "Result".

Ngoài ra, chúng ta sẽ thực hiện thêm những thao tác khác đối với level 3. Các hàm thao tác với level 3 đã được đề cập ở mục **2.9.3**.

Tương tự với level 3, level 4 cũng sẽ có các thao tác riêng. Các hàm thao tác với Level 4 đã được đề cập ở mục **2.9.4**.

2.10 Nhóm hàm khai báo trong file HighScore.h

• Hàm `DrawScoreBoard()`: thực hiện chức năng vẽ bảng chứa thông tin 15 người chứa điểm cao nhất được lưu trong file highscore.txt

```
1 int DrawScoreBoard(int cornerX, int cornerY, int width, int height) {
```

```

  system("cls");

  fstream file;
  string Name;
  char date[11];
  int levEl, scOre;

  file.open("highscore.txt", ios_base::in | ios_base::out);
  file.seekg(0, ios_base::end);
  int x = file.tellg();
  x = x / 29;

  //emty file ?
  if (x == 0) {
    DrawBoard(cornerX, cornerY, width, height);
    printASCII("highscoreText.txt", cornerX + 9, cornerY - 4);
    printASCII("highscoreMess.txt", 1, 7);
    GotoXY(60, 14); cout << "Nothing here :(";
    GotoXY(55, 15); cout << "Press ESC to  back to MENU";
    file.close();
    char ch1;
    while (1) {
      if (_kbhit()) {
        ch1 = _getch();
        if (ch1 == 27) {
          PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
          break;
        }
      }
    }
    system("cls");
    return 1;
  }

  TextColor(6);
  GotoXY(cornerX + 5, cornerY + 2);
  int space = width / 5 - 1;
  cout << setw(space + 1) << left << "Top"
    << setw(space) << left << "Name"
    << setw(space + 3) << left << "Score"
    << setw(space) << left << "Lv"
    << setw(space) << left << "Date";

  if (x == 16) x = 15;
  file.seekg(0, ios_base::beg);
  for (int i = 0; i < x; i++) {
    file >> Name >> scOre >> levEl;
    file.seekg(2, ios_base::cur);
    file.getline(date, 11);
    GotoXY(cornerX + 5, cornerY + 4 + i);
    cout << setw(space) << left << i + 1
      << setw(space + 2) << left << Name
      << setw(space + 3) << left << scOre
      << setw(space - 2) << left << levEl
      << setw(space) << left << date;
  }
  file.close();
  // draw frame of load board
  TextColor(0);
```

```
62    int c = 0;
63    int space1 = width / 5;
64    DrawBoard(cornerX, cornerY, width, height);
65    DrawBoard(cornerX, cornerY, width, 3);
66    do
67    {
68      c += space1;
69      DrawBoard(cornerX + c, cornerY, width / 5, height);
70    } while (c < WIDTH_CONSOLE / 2);
71
72    GotoXY(cornerX, cornerY + 3); cout << char(204);
73    GotoXY(cornerX + width, cornerY + 3); cout << char(185);
74    c = 1;
75    while (c <= 4)
76    {
77      GotoXY(cornerX + c * space1, cornerY); cout << char(203);
78      GotoXY(cornerX + c * space1, cornerY + 3); cout << char(206);
79      GotoXY(cornerX + c * space1, cornerY + height); cout << char(202);
80      c++;
81    }
82    TextColor(0);
83    printASCII("highscoreText.txt", cornerX + 9, cornerY - 4);
84    printASCII("highscoreMess.txt", 1, 7);
85    char ch1;
86    while (1) {
87      if (_kbhit()) {
88        ch1 = _getch();
89        if (ch1 == 27) {
90          PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
91          break;
92        }
93        else if (ch1 == 'q' || ch1 == 'Q') {
94          PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
95          clearfile("highscore.txt");
96          break;
97        }
98      }
99    }
100   system("cls");
101   return 0;
102 }
```

Mở file, tính số dòng trong file highscore.txt (dòng 10 đến 13), kiểm tra file rỗng (dòng 16 đến 35), nếu file rỗng, vẽ bảng đề nghị người chơi ấn phím esc để quay lại, trả về 1.

Nếu file không rỗng, in ra màn hình thông tin người chơi đã lưu điểm, tối đa 15 người (dòng 37 đến 58), vẽ bảng và trang trí (dòng 60 đến 84), chờ người chơi ấn phím esc để quay lại hoặc phím Q để xóa dữ liệu và quay lại (dòng 86 đến 99), trả về 0.

• Hàm `SaveHighScore()`: thực hiện chức năng lưu tên, điểm, màn chơi và thời điểm hiện tại vào file highscore.txt

```
1  void SaveHighScore(string nAme) {
2    int x = 0; // number of line
3    fstream file;
4    file.open("highscore.txt", ios_base::in | ios_base::out);
5    // line check
6    file.seekg(0, ios_base::end);
7    x = file.tellg();
8    x = x / 29;
9    //
```

```
10    if (x >= 16) {
11      file.seekg(-29, ios_base::end);
12      file << setw(27) << " ";
13      file.seekg(-27, ios_base::cur);
14    }
15    file << setw(11) << left << nAme << setw(3) << left << SCORE << setw(3) <<
        left << level;
16
17    time_t now = time(0);
18    tm* ltm = localtime(&now);
19
20    file << setw(2) << right << ltm->tm_mday << "/";
21    file << setw(2) << right << 1 + ltm->tm_mon << "/";
22    file << setw(4) << right << 1900 + ltm->tm_year;
23    file << "\n";
24    // 29 byte per line
25    file.close();
26  }
```

Mở file, tính số dòng trong file (dòng 2 đến 8). Ghi vào file highscore.txt (dòng 10 đến 24). Nếu số dòng nhỏ hơn hoặc bằng 15, ghi các thông tin theo thứ tự: tên, điểm, màn chơi, thơi gian lưu điểm; nếu số dòng lớn hơn 15, dịch con trỏ 29 byte (độ dài dòng gồm kí tự xuống dòng) từ dòng cuối cùng, tạo khoảng trắng dài 27 byte (độ dài dòng không gồm kí tự xuống dòng) (xóa dòng 16 trong file), ghi vào dòng 16 các thông tin theo thứ tự: tên, điểm, màn chơi, thời gian lưu.

• Hàm `highscore_structIn()`: Hàm nhận vào đối số là một mảng struct. Hàm thực hiện chức năng tải lên các thông tin trong file highscore.txt vào mảng struct HighScore

```
1  void highscore_structIn(HighScore x[]) {
2    fstream file;
3    file.open("highscore.txt", ios_base::in | ios_base::out);
4    int nLine = 0;
5    file.seekg(0, ios_base::end);
6    nLine = file.tellg();
7    nLine = nLine / 29;
8    file.seekg(0, ios_base::beg);
9    for (int i = 0; i < nLine; i++) {
10     file >> x[i].nAme >> x[i].score >> x[i].lv;
11     file.seekg(2, ios_base::cur);
12     file.getline(x[i].dAte, 11);
13   }
14   file.close();
15 }
```

Mở file highscore.txt, tính số dòng (dòng 2 đến 7), dùng vòng lặp for để gán các trường của struct bằng các thông tin trong file highscore.txt (dòng 9 đến 13).

• Hàm `highscoreSort()`: Hàm thực hiện sắp xếp mảng theo số điểm từ cao đến thấp

```
1  void highscoreSort(HighScore x[]) {
2    HighScore temp;
3    fstream file;
4    file.open("highscore.txt", ios_base::in | ios_base::out);
5    int nLine = 0;
6    file.seekg(0, ios_base::end);
7    nLine = file.tellg();
8    nLine = nLine / 29;
9    for (int i = 0; i < nLine - 1; i++) {
10     for (int j = i + 1; j < nLine; j++) {
11       if (x[i].score < x[j].score) {
```

```
12          temp = x[i];
13          x[i] = x[j];
14          x[j] = temp;
15        }
16      }
17    }
18    file.seekg(0, ios_base::beg);
19    for (int i = 0; i < nLine; i++) {
20      file << setw(11) << left << x[i].nAme << setw(3) << left << x[i].score
       << setw(3) << left << x[i].lv << x[i].dAte << "\n";
21    }
22    file.close();
23  }
```

Mở file, tính số dòng (dòng 3 đến 8), dùng 2 vòng for để so sánh theo số điểm (dòng 9 đến 17), trả con trỏ về đầu file, dùng for để ghi vào file các thông tin đã sắp xếp (dòng 18 đến 21).

• Hàm `hs()`: thực hiện tổng hợp 3 hàm SaveHighScore, highscore_structIn, highscoreSort.

```
1  void hs(string nAme, HighScore x[]) {
2    SaveHighScore(nAme);
3    highscore_structIn(x);
4    highscoreSort(x);
5  }
```

• Hàm `getNameforHighScore()`: Thực hiện vẽ bảng dùng để nhập tên nếu người dùng muốn lưu điểm.

```
1  string getNameforHighScore() {
2    ClearBoard();
3    TextColor(9);
4    printASCII("savegame.txt", cornerX + 6, cornerY + 2);
5    TextColor(0);
6
7    GotoXY(cornerX + 44, cornerY + 7); cout << (char)201;
8    GotoXY(cornerX + 69, cornerY + 7); cout << (char)187;
9    GotoXY(cornerX + 44, cornerY + 17); cout << (char)200;
10   GotoXY(cornerX + 69, cornerY + 17); cout << (char)188;
11   for (int i = 1; i < 25; i++)
12   {
13     GotoXY(cornerX + 44 + i, cornerY + 7); cout << (char)205;
14     GotoXY(cornerX + 44 + i, cornerY + 17); cout << (char)205;
15   }
16   for (int i = 1; i < 10; i++)
17   {
18     GotoXY(cornerX + 44, cornerY + 7 + i); cout << (char)186;
19     GotoXY(cornerX + 69, cornerY + 7 + i); cout << (char)186;
20   }
21   GotoXY(cornerX + 46, cornerY + 9); cout << "       ONLY";
22   GotoXY(cornerX + 46, cornerY + 10); cout << "alphanumeric characters";
23   GotoXY(cornerX + 46, cornerY + 12); cout << "      A - Z (a - z)   ";
24   GotoXY(cornerX + 46, cornerY + 14); cout << "      0-9     ";
25   GotoXY(cornerX + 46, cornerY + 16); cout << "[At most 10 characters]";
26   GotoXY(cornerX + 15, cornerY + 9); cout << "Enter your name: ";
27   GotoXY(cornerX + 10, cornerY + 12); cout << "=>";
28   cin.clear();
29   string nAme;
30   do {
31     GotoXY(cornerX + 16, cornerY + 12); cout << "          ";
32     GotoXY(cornerX + 16, cornerY + 12);
33     cin.clear();
```

```
34      nAme = LimInput(MAX_NAME);
35    } while (NameCheck(nAme, "highscore.txt"));
36    for (int i = cornerX + 6; i < cornerX + 6 + 60; i++)
37    {
38      for (int j = cornerY + 2; j < cornerY + 2 + 5; j++)
39      {
40        GotoXY(i, j); cout << " ";
41      }
42
43    }
44    TextColor(11);
45    printASCII("saved.txt", cornerX + 20, cornerY + 2);
46    TextColor(0);
47    GotoXY(cornerX + 10, cornerY + 12); cout << "Press ANY KEY to  back to
       MENU";
48    char xx = _getch();
49    return nAme;
50  }
```

Vẽ và trang trí bảng (dòng 2 đến 27), thực hiện nhập tên và kiểm tra tên (dòng 30 đến 35), nếu nhập tên trùng, xóa tên đã nhập trên màn hình và tiến hành thông báo cho người chơi nhập lại. Nếu nhập không trùng, vẽ bảng thông báo cho người chơi đã nhập và lưu thành công (dòng 36 đến 48), trả về tên đã nhập (dòng 49).

• Hàm `NameCheck()`: Thực hiện chức năng so sánh tên cần kiểm tra với các tên có sẵn trong file truyền vào.

```
1  bool NameCheck(string naMe, string filename) {
2    fstream file;
3    file.open(filename, ios_base::in);
4    string NaMe;
5    int line = 0;
6    file.seekg(0, ios_base::end);
7    line = file.tellg();
8    if (filename == "highscore.txt") line = line / 29;
9    else if (filename == "SaveInf.txt") line = line / 32;
10
11   file.seekg(0, ios_base::beg);
12   for (int i = 0; i < line; i++) {
13     file >> NaMe;
14     if (NaMe == naMe) {
15       file.close();
16       TextColor(rand() % 5 + 2);
17       GotoXY(cornerX + 13, cornerY + 14); cout << "Name already exists";
18       TextColor(0);
19       return 1;
20     }
21     if (filename == "highscore.txt") file.seekg(29 * (i + 1), ios_base::beg)
       ;
22     else if (filename == "SaveInf.txt") file.seekg(32 * (i + 1), ios_base::
       beg);
23   }
24   GotoXY(cornerX + 13, cornerY + 14); cout << "                    ";
25   file.close();
26   return 0;
27 }
```

Mở file, tính số dòng đối với từng file khác nhau (dòng 2 đến 9), dịch con trỏ về đầu file, dùng vòng lặp for vơi biến i từ 0 đến số dòng tối đa, so sánh tên đã có sẵn (dòng 12 đến 23). Trong vòng lặp for (dòng 21, 22), sau mỗi lần so sánh, nếu không trùng thì con trỏ sẽ được

dịch một đoạn i + 1 nhân cho độ dài mỗi dòng của các file tương ứng – dịch từ đầu file (hay có nghĩa là dịch con trỏ đến đầu dòng tiếp theo để so sánh).

• Hàm `LimInput()`: Hàm giới hạn số kí tự nhập

```
char* LimInput(const int k) {
  char* Out = new char[k + 1];
  int n, index = 0;
  do {
  loop:;
    n = _getch();
    if (((n >= '0' && n <= '9') || (n >= 'A' && n <= 'Z') || (n >= 'a' && n
   <= 'z') || n == '.') && index < k)
    {
      cout << char(n);
      Out[index++] = n;
    }
    else if (n == '\b' && index > 0)
    {
      cout << "\b \b";
      Out[--index] = 0;
    }
    if (n == 13 && index == 0) goto loop;
  } while (n != 13);
  Out[index] = 0;
  return Out;
}
```

Mã nguồn của hàm tham khảo từ [1]

2.11 Nhóm hàm khai báo trong file SaveLoad.h

• Hàm `Save()`: hực hiện chức năng lưu trữ dự liệu game vào file Save.txt

```
void Save() {
  fstream file;
  file.open("Save.txt", ios_base::in | ios_base::out);

  file.seekg(0, ios_base::end);
  file << setw(800) << " ";
  file.seekg(-800, ios_base::cur);

  //level
  file << level << " ";
  //score
  file << SCORE << " ";
  // snake
  file << SIZE_SNAKE << " " << SIZE_SNAKE_1 << " ";
  for (int i = SIZE_SNAKE - 1; i >= 0; i--) {
    file << snake[i].x << " " << snake[i].y << " ";
  }
  // food
  file << FOOD_INDEX << " ";
  for (int i = 0; i < MAX_SIZE_FOOD - 1; i++) {
    file << food[i].x << " " << food[i].y << " ";
  }
  // gate
  for (int i = 0; i < 5; i++) {
    file << gate[i].x << " " << gate[i].y << " ";
```

```
26    }
27    file << GATE << " ";
28    // moving
29    file << MOVING << " " << CHAR_LOCK << " ";
30    // speed
31    file << SPEED << " ";
32    // sth else
33    file << flag << " " << outgate << " " << fix << " " << fixcount << " " <<
       play_again << " ";
34    // lv 3
35    if (level == 3) {
36      // object
37      for (int i = 0; i < 4; i++) {
38        file << object[i].x << " " << object[i].y << " ";
39      }
40      // sth else
41      file << object_state << " "
42        << object_state1 << " "
43        << object_state2 << " "
44        << object_state3 << " "
45        << eated << " "
46        << stoptime << " "
47        << supercount << " "
48        << undying << " "
49        << supercount1 << " "
50        << object_fix << " "
51        << object_fix1 << " ";
52      for (int i = 0; i < MAX_SUPER_FOOD; i++) {
53        file << superfood_eated[i] << " ";
54      }
55      for (int i = 0; i < 5; i++) {
56        file << superfood[i].x << " " << superfood[i].y << " ";
57      }
58    }
59    //lv 4
60    else if (level == 4) {
61      //ball
62      for (int i = 0; i < 2; i++) {
63        file << ball[i].x << " " << ball[i].y << " ";
64      }
65      file << ball_state << " ";
66      //smallwall_hitted
67      for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++) {
68        file << smallwall_hitted[i] << " ";
69      }
70      //smallwall_hitted_idx & goalgoalgoal & length_break
71      file << smallwall_hitted_idx << " " << goalgoalgoal << " " <<
       length_break;
72    }
73    file.seekg(0, ios_base::end);
74    file << "\n";
75    file.close();
76  }
```

Đầu tiên thực hiện thao tác mở file (dòng 2, 3), đặt con trỏ về cuối file, tạo ra 800 khoản cách và trả về vị trí trước khoản cách đầu tiên (dòng 5, 6, 7). Lần lượt lưu các thông số của game vào file (mỗi thông số cách nhau một dấu cách) theo thứ tự: level, SCORE, SIZE_SNAKE, SIZE_SNAKE_1, snake[], FOOD_INDEX, food[], gate[], GATE, MOVING, CHAR_LOCK, SPEED. flag, outgate, fix, fixcount, play_again. Nếu đang ở màn 3 hoặc 4 sẽ lưu thêm các thông số: object[], object_state, object_state1, object_state2, object_state3, eated, stop-

time, supercount, undying, supercount1, object_fix, object_fix1, superfood_eated[], superfood[] (đối với màn 3); ball[], ball_state, smallwall_hitted[], smallwall_hitted_idx, goagoalgoal, length_break (đối với màn 4); sau đỏ trả con trỏ về cuối file và xuống dòng.

• Hàm `LoadFile()`: Thực hiện chức năng đọc dữ liệu và lưu vào các biến cần thiết của dòng được truyền vào hàm.

```
void LoadFile(int n) {
  int distance = n * 802;
  ifstream filein;
  filein.open("Save.txt", ios_base::in);
  filein.seekg(-distance, ios_base::end);
  filein.seekg(1, ios_base::cur);

  filein >> SCORE >> SIZE_SNAKE >> SIZE_SNAKE_1;

  for (int i = SIZE_SNAKE - 1; i >= 0; i--) {
    filein >> snake[i].x >> snake[i].y;
  }
  filein >> FOOD_INDEX;
  for (int i = 0; i < MAX_SIZE_FOOD - 1; i++) {
    filein >> food[i].x >> food[i].y;
  }
  for (int i = 0; i < 5; i++) {
    filein >> gate[i].x >> gate[i].y;
  }
  filein >> GATE >> MOVING >> CHAR_LOCK >> SPEED
    >> flag >> outgate >> fix >> fixcount >> play_again;
  if (level == 3) {
    for (int i = 0; i < 4; i++) {
      filein >> object[i].x >> object[i].y;
    }
    filein >> object_state
      >> object_state1
      >> object_state2
      >> object_state3
      >> eated
      >> stoptime
      >> supercount
      >> undying
      >> supercount1
      >> object_fix
      >> object_fix1;
    for (int i = 0; i < MAX_SUPER_FOOD; i++) {
      filein >> superfood_eated[i];
    }
    for (int i = 0; i < 5; i++) {
      filein >> superfood[i].x >> superfood[i].y;
    }
  }
  else if (level == 4) {
    for (int i = 0; i < 2; i++) {
      filein >> ball[i].x >> ball[i].y;
    }
    filein >> ball_state;

    for (int i = 0; i < MAX_SIZE_SMALL_WALL; i++) {
      filein >> smallwall_hitted[i];
    }

```

```
54      filein >> smallwall_hitted_idx >> goalgoalgoal >> length_break;
55    }
56    filein.close();
57  }
```

Đầu tiên thực hiện tính <khoảng cách> để đưa con trỏ về đâu dòng cần đọc dữ liệu bằng cách lấy số dòng được đưa vào hàm nhân cho 802 (độ dài mỗi dòng trong file Save.txt (bao gồm kí tự xuống dòng)) (dòng 2). Thực hiện mở file Save.txt, dịch chuyển con trỏ một <khoảng cách> bắt đầu từ cuối file, dịch tiếp còn trỏ sang phải 1 byte (bỏ qua thông số level trong file) (dòng 3, 4, 5, 6), đọc tất cả các thông số được lưu trong dòng đã chọn (dòng 8 đến dòng 54).

• Hàm `SaveInf()`: Thực hiện chức năng lưu thông tin người chơi vào file SaveInf.txt

```
1  void SaveInf(char* nAme) {
2    fstream file;
3    file.open("SaveInf.txt", ios_base::in | ios_base::out);
4    file.seekg(0, ios_base::end);
5
6    file << setw(11) << left << nAme;
7    file << setw(3) << left << level;
8    file << setw(5) << left << SCORE;
9
10   time_t now = time(0);
11   tm* ltm = localtime(&now);
12
13   file << setw(2) << right << ltm->tm_mday << setw(1) << "/";
14   file << setw(2) << right << 1 + ltm->tm_mon << setw(1) << "/";
15   file << setw(5) << left << 1900 + ltm->tm_year << "\n";
16
17   // 30 byte/line (+ \n = 32)
18   file.close();
19 }
```

Mở file SaveInf.txt, dịch con trỏ về cuối file (dòng 2, 3, 4), ghi vào file các thông tin gồm: nAme, level, SCORE (dòng 6, 7, 8), tính thời điểm hiện tại và ghi vào file (dòng 10 đến dòng 15).

• Hàm `LvLoad()`: Thực hiện chức năng đọc màn chơi đã chọn

```
1  void LvLoad(int n) {
2    ifstream file;
3    int distance = n * 802;
4    int lV;
5    file.open("Save.txt", ios_base::in | ios_base::out);
6    file.seekg(-distance, ios_base::end);
7    file >> lV;
8    level = lV;
9    file.close();
10 }
```

Tính <khoảng cách> tương tự như hàm LoadFile() bằng cách lấy số thứ tự dòng nhân cho 802, dịch con một <khoảng cách> từ cuối file, đọc màn chơi (số được viết đầu dòng) và lưu vào biến level.

• Hàm `DrawSaveInf()`: Thực hiện chức năng in ra màn hình các thông tin trong file SaveInf.txt từ dòng bắt đầu đến dòng kết thúc

```
1  void DrawSaveInf(int start, int end, int space) {
2    fstream file;
3    file.open("SaveInf.txt", ios_base::out | ios_base::in);
```

```
  string Name;
  char date[11];
  int levEl, scOre;
  int j = 0;
  for (int i = start; i < end; i++) {
    file.seekg(-32 * (static_cast<std::basic_istream<char, std::char_traits<
    char>>::off_type>(i) + 1), ios_base::end);
    file >> Name >> levEl >> scOre;
    if (scOre < 10)  file.seekg(4, ios_base::cur);
    else if (scOre < 100 && scOre >= 10) file.seekg(3, ios_base::cur);
    else if (scOre < 1000 && scOre >= 100) file.seekg(2, ios_base::cur);
    else file.seekg(1, ios_base::cur);
    file.getline(date, 11);
    file.clear();
    TextColor(6);
    GotoXY(cornerX + 7, cornerY + 5 + j); cout << setw(10) << Name;
    GotoXY(cornerX + 7 + space * 1, cornerY + 5 + j); cout << setw(2) <<
    levEl;
    GotoXY(cornerX + 7 + space * 2, cornerY + 5 + j); cout << setw(5) <<
    scOre;
    GotoXY(cornerX + 7 + space * 3 - 6, cornerY + 5 + j); cout << setw(11)
    << date << endl;
    j++;
  }
  file.close();
}
```

Mở file SaveInf.txt, dùng vòng lặp for để đọc: với mỗi biến i tăng từ dòng bắt đầu đến dòng kết thúc, dịch con trỏ một khoản 32 byte (độ dài cố định của mỗi dòng trong file SaveInf.txt) nhân cho i từ vị trí cuối file, đọc các thông tin trong dòng và thực hiện in ra màn hình.

• Hàm `DrawLoadBoard()`: Hàm thực hiện chức năng vẽ bảng chứa các thông tin được lưu trong trong file SaveInf.txt, vẽ menu để chọn một trong các thông tin được lưu trong mỗi dòng trong file SaveInf.txt và trả về số thứ tự của dòng có chứa thông tin (từ dưới lên).

```
void LvLoad(int n) {
  ifstream file;
  int distance = n * 802;
  int lV;
  file.open("Save.txt", ios_base::in | ios_base::out);
  file.seekg(-distance, ios_base::end);
  file >> lV;
  level = lV;
  file.close();
}

int DrawLoadBoard(int cornerX, int cornerY, int width, int height) {
  system("cls");

  fstream file;
  file.open("SaveInf.txt", ios_base::in | ios_base::out);

  int nLine = 0;
  file.seekg(0, ios_base::end);
  nLine = file.tellg();
  nLine = nLine / 32;

  if (nLine == 0) {
    DrawBoard(cornerX, cornerY, width, height);
    printASCII("load.txt", cornerX + 25, cornerY - 5);
```

```
    printASCII("messageLoad.txt", 2, 2);
    GotoXY(60, 14); cout << "Nothing here :(";
    GotoXY(55, 15); cout << "Press ESC to  back to MENU";
    file.close();
    return -1;
  }
  GotoXY(cornerX + 9, cornerY + 2);
  int space = width / 4 + 2;
  GotoXY(cornerX + 7, cornerY + 2);
  TextColor(6);
  cout << setw(space) << left << "Name" << setw(space - 2) << left << "Lv"
   << setw(space - 1) << left << "Score" << setw(space + 100) << left << "
   Date";
  GotoXY(cornerX + 3, cornerY + 4);

  if (nLine < 15) {
    DrawSaveInf(0, nLine, space);
  }
  else {
    DrawSaveInf(0, 15, space);
  }

  TextColor(0);
  DrawBoard(cornerX, cornerY, width, height);

  int c = 0;
  int space1 = width / 4 + 1;
  DrawBoard(cornerX, cornerY, width, 3);
  do
  {
    c += space1;
    DrawBoard(cornerX + c, cornerY, space1, height);

  } while (c < WIDTH_CONSOLE / 3);

  GotoXY(cornerX, cornerY + 3); cout << char(204);
  GotoXY(cornerX + width, cornerY + 3); cout << char(185);

  c = 1;
  int d = 0;
  while (c <= 3)
  {
    GotoXY(cornerX + c * space1, cornerY); cout << char(203);
    GotoXY(cornerX + c * space1, cornerY + 3); cout << char(206);
    GotoXY(cornerX + c * space1, cornerY + height); cout << char(202);
    c++;
  }
  //menu-----------------
  int i = 1;
  int x = cornerX + 2, y = cornerY + 5, xp = x, yp = y;
  int oldX = 0, oldY = 0;
  bool k = true;
  int mau = 1;

  printASCII("load.txt", cornerX + 25, cornerY - 5);
  printASCII("messageLoad.txt", 2, 2);

  while (1) {
    if (k) {
      GotoXY(oldX, oldY); cout << "  ";
```

```
84        GotoXY(xp, yp); cout << ">>";
85        oldX = xp; oldY = yp;
86        k = false;
87      }
88      if (_kbhit()) {
89        char ch = toupper(_getch());
90        k = true;
91        // draw "LOAD" and "message.txt" with color
92        if (mau > 16)
93        {
94          mau = 1;
95        }
96        if (mau % 7 == 0 || mau % 15 == 0)
97        {
98          mau += 2;
99        }
100       TextColor(mau++);
101       printASCII("load.txt", cornerX + 25, cornerY - 5);
102       printASCII("messageLoad.txt", 2, 2);
103       TextColor(TEXT_COLOR);
104       if (ch == 27)
105       {
106         menuOn = 1;
107         return 0;
108       }
109       else if (ch == 'Q') {
110         PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
111         clearfile("Save.txt");
112         clearfile("SaveInf.txt");
113         menuOn = 1;
114         return 0;
115       }
116       if (ch == 'W') {
117         if (yp != y) {
118           yp--;
119           i--;
120         }
121         else if (i > 1) {
122           i--;
123           DrawSaveInf(i - 1, i + 14, space);
124         }
125       }
126       else if (ch == 'S') {
127         if (nLine < 15) {
128           if (yp != nLine - 1 + y) {
129             yp++;
130             i++;
131           }
132           else if (i < nLine) {
133             i++;
134             DrawSaveInf(i - 15, i, space);
135           }
136         }
137         else {
138           if (yp != 14 + y) {
139             yp++;
140             i++;
141           }
142           else if (i < nLine) {
143             i++;
```

```
144             DrawSaveInf(i - 15, i, space);
145           }
146         }
147       }
148       if (ch == 32) {
149         PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
150         return i;
151       }
152     }
153   }
154   return -1;
155 }
```

Mở file SaveInf.txt, tính số dòng (đưa con trỏ về cuối file, tính số byte vị trí con trỏ đang ở, chia cho 32 (byte) (độ dài cố định mỗi dòng, chứa kí tự xuống dòng) (dòng 4 đến dòng10), nếu file rỗng: vẽ khung và trả về -1 (dòng 12 đến 20).

Nếu file không rỗng, vẽ khung và vẽ các thông tin trong file SaveInf.txt (dùng hàm DrawSaveInf, DrawBoard,... đã được đề cập phía trên) (dòng 21 đến 59).

Vẽ menu (dòng 70 đến 144), dùng vòng lặp while, mỗi khi ấn phím W hoặc S, dấu » trên màn hình sẽ di chuyển lên hoặc xuống, ấn phím cách để trả về vị trí của dấu » đang ở chính là vị trí của dòng cần chọn trong file SaveInf.txt.

• Hàm `LoadGame()`: Hàm thực hiện chức năng kiểm tra chơi tiếp hay chơi lại từ đầu, nếu chơi tiếp, vẽ bảng các thông tin đã lưu để người chơi chọn và tiến hành tải lên các thông tin để chơi tiếp, nếu không thì chơi lại từ đầu. Hàm nhận vào 1 đối số kiểu int, nếu bằng 1 thì chơi tiếp, nếu không thì chơi lại từ đầu

```
1  void LoadGame(int n) {
2    if (n == 1) {
3      int choice = DrawLoadBoard(cornerX, cornerY, WIDTH_CONSOLE,
     HEIGH_CONSOLE);
4      if (choice == 0) return;
5      else if (choice == -1) {
6        int ch;
7        while (1) {
8          if (_kbhit()) {
9            ch = _getch();
10           if (ch == 27) {
11             PlaySound(TEXT("selectclick.wav"), NULL, SND_SYNC);
12             menuOn = 1;
13             break;
14           }
15         }
16       }
17       return;
18     }
19     else {
20       LvLoad(choice);
21       DrawInfo(corLx, corLy + 1);
22       if (level == 1) {
23         StartGame();
24       }
25       else if (level == 2) {
26         StartGame1();
27       }
28       else if (level == 3) {
29         StartGame2();
30       }
31       else if (level == 4)
```

```
32        {
33          StartGame3();
34        }
35        LoadFile(choice);
36
37        if (level == 2) {
38          ClearBoard();
39          DrawCountDown(cornerX + 33, cornerY + 7);
40          DrawWall();
41        }
42        else if (level == 3) {
43          DrawCountDown(cornerX + 33, cornerY + 7);
44          Object();
45          if (FOOD_INDEX >= 2 && FOOD_INDEX <= 3)
46          {
47            DoubleObject();
48          }
49          if (FOOD_INDEX == 4)
50          {
51            SingleObject(object_state3, 3);
52            SingleObject(object_state2, 2);
53            SingleObject(object_state1, 1);
54          }
55        }
56        else {
57          DrawCountDown(cornerX + 33, cornerY + 7);
58        }
59
60        if (level == 4 && (GATE == 1 || goalgoalgoal == 1)) {
61          if (GATE == 1) goal_celebration = 16;
62          DelWall();
63        }
64      }
65    }
66    else {
67      StartGame();
68      play_again = 1;
69      SPEED = 1;
70      DrawCountDown(cornerX + 33, cornerY + 7);
71    }
72  }
```

Nếu truyền vào 1, hàm thực hiện vẽ bảng lựa chọn bằng hàm DrawLoadBoard (dòng 3), nếu DrawLoadBoard trả về 0 hoặc trả về -1 (file rỗng) quay lại menu mở đầu (dòng 4 đến 18) ; nếu không, thực hiện tải các thông tin đã lưu và tiếp tục chơi (dòng 19 đến 64). Nếu truyền vào số khác 1, hàm thực hiện trò chơi mới (dòng 67 đến 70).

• Hàm `Clearfile()`: Nhận vào đối số kiểu string là tên file cần xóa dữ liệu. Hàm có chức năng xóa dữ liệu trong file bằng cách mở file để ghi và cắt sau đó đóng file

```
1  void clearfile(string filename) {
2    ofstream file;
3    file.open(filename, ofstream::out | ofstream::trunc);
4    file.close();
5  }
```

2.12 Hàm main() trong file snake0.cpp

```
int main() {
  int temp;
  FixConsoleWindow();
  DisableSelection();
  Nocursortype();
  BackGroundColor(BACK_GROUND_COLOR);
  TextColor(TEXT_COLOR);

  thread t2(Sound); //sound
  gameIntro();

  while (true)
  {
    ResetGlobal();
    MenuGame();

    thread t1(ThreadFunc); //Create thread for snake
    HANDLE handle_t1 = t1.native_handle(); //Take handle of thread
    while (1) {
      temp = toupper(_getch());
      if (STATE == 1) {
        if (temp == 'P') {
          PauseGame(handle_t1);
          if (save_load == 1)
          {
            if (t1.joinable())
            {
              t1.detach();
            }
            break;
          }
        }
        else if (temp == 27) { // esc
          SuspendThread(handle_t1);
          Exit();
        }
        else {
          ResumeThread(handle_t1);
          if ((temp != CHAR_LOCK) && (temp == 'D' || temp == 'A' || temp ==
    'W' || temp == 'S'))
          {
            if (temp == 'D') CHAR_LOCK = 'A';
            else if (temp == 'W') CHAR_LOCK = 'S';
            else if (temp == 'S') CHAR_LOCK = 'W';
            else CHAR_LOCK = 'D';
            MOVING = temp;
          }
        }
      }
      else {
        SuspendThread(handle_t1);
        if (temp == 'E')
        {
          if (t1.joinable())
          {
            t1.detach();
          }
          break;
        }
        else if (temp == 'S') {
```

```
60            hs(getNameforHighScore(), H);
61            if (t1.joinable())
62            {
63              t1.detach();
64            }
65            break;
66          }
67        }
68      }
69    }
70    return 0;
71  }
```

Cuối cùng, ta xây dựng hàm main để kết nối các hàm đã được viết. Đầu tiền ta gọi các hàm "FixConsoleWindow", "DisableSelection", "Nocursortype" lần lượt để điều chỉnh kích thước màn hình console, ngăn không cho người dùng thao tác với chuột trên màn hình console và ẩn con trỏ nháy trong màn hình console. Sau đó ta gọi hàm "BackGroundColor" và "TextColor" để điều chỉnh màu của background và màu của kí tự. Tiếp theo, ta tạo một thread sử dụng hàm "Sound" để tạo thực hiện các thao tác với âm thanh trước khi gọi hàm "gameIntro" để tạo phần mở đầu cho game. Sau đó ta sẽ bước vào vòng lặp chính của game.

Đàu tiên, ta gọi hàm "ResetGlobal" để khởi tạo lại một số biến toàn cục bị thay đổi trong quá trình chơi trước khi chơi lại. Kế tiếp, ta gọi hàm "MenuGame" để tạo Menu , cho phép người dùng thao trực tiếp với menu của game. Ta sẽ thoát khỏi hàm này chỉ khi người dùng bắt đầu màn chơi mới, load lại màn chơi cũ hoặc thoát chương trình. Ta tiếp tục tạo thread sử dụng hàm "ThreadFunc" và ta khởi tạo "HANDLE" cho thread để dùng cho hàm "PauseGame". Sau đó ta bước vào vòng lặp của một màn chơi.

Đầu tiên ta cho phép người dùng bấm bàn phím. Sau đó ta xét nếu STATE = 1, ta xét xem người dùng đã bấm phím gì. Đối với các phím di chuyển, ta liên tục cập nhật cho biến "CHAR_LOCK" giá trị ngược với hướng di chuyển mà người dùng đã bấm. Trong trường hợp người dùng bấm phím "P", ta gọi hàm "PauseGame" để dừng game lại, sau đó kiểm tra biến "save_load" để xem người dùng có thực hiện lưu game lại hay không. Nếu có, ta thực hiện "detach" thread trước khi "break" để thoát vòng lặp. Nếu người dùng bấm phím "E", ta dừng thread bằng "SuspendThread" trước khi gọi hàm "Exit" để thoát game.

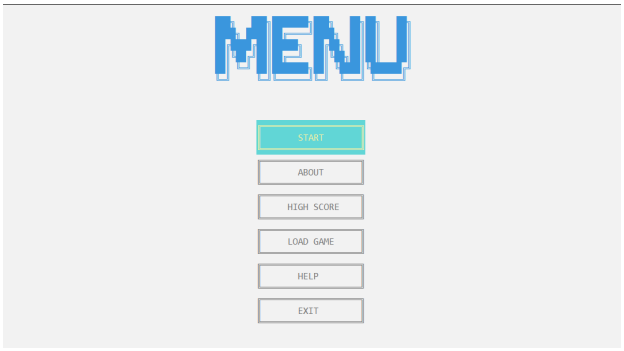
Trong trường hợp STATE = 0, ta dừng thread bằng hàm "SuspendThread". Sau đó ta xét người dùng bấm phím gì (lúc này đang hiển thị màn hình kết quả). Nếu người dùng bấm phím "E" ta cũng thực hiện "detach" thread và "break", nếu người dùng bấm phím "S" ta gọi hàm "hs" để lưu lại kết quả của người chơi trước khi "detach" thread và "break".

CHƯƠNG 3

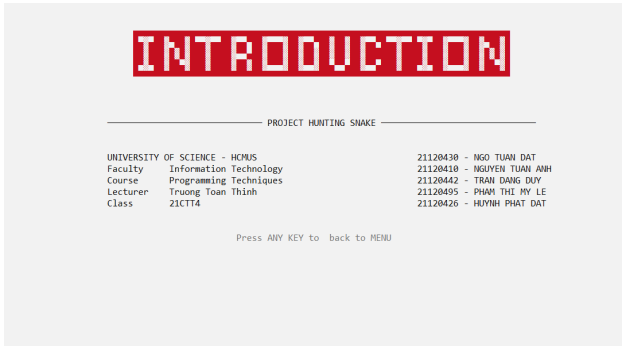
Kết quả

Chương này cung cấp một số hình ảnh của chương trình khi đã chạy thành công

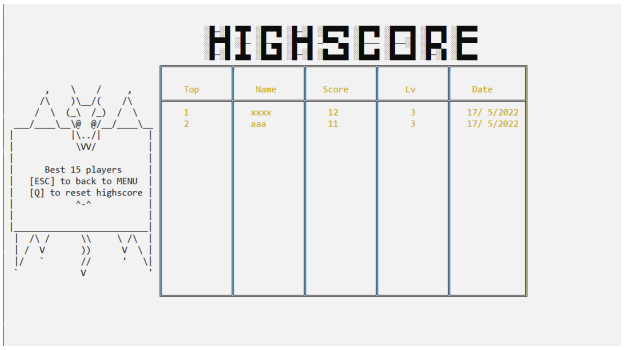
Hình 3.1: Giao diện mở đầu



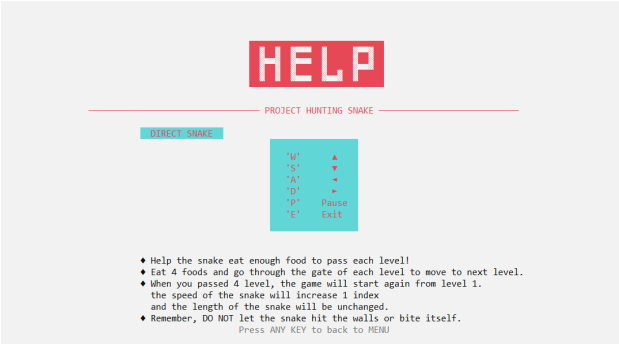
Hình 3.2: Giao diện Menu



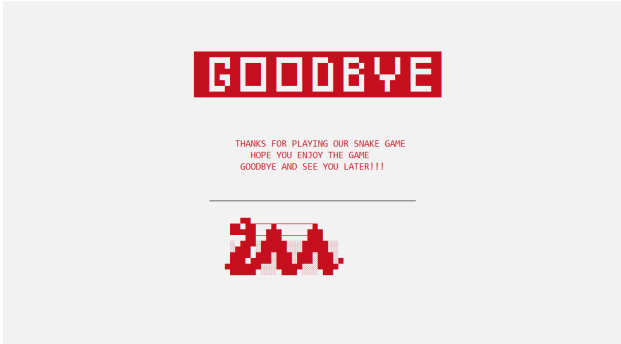
Hình 3.3: Giới thiệu thành viên



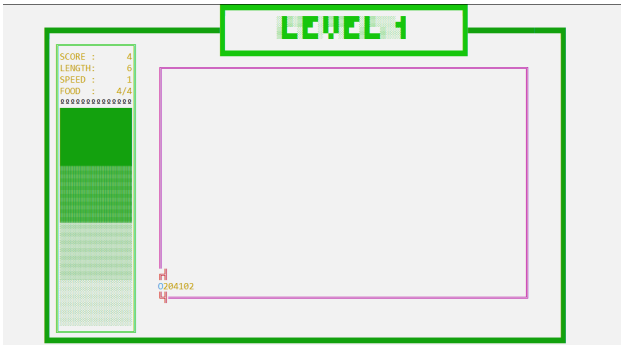
Hình 3.4: Bảng người chơi có số điểm cao nhất



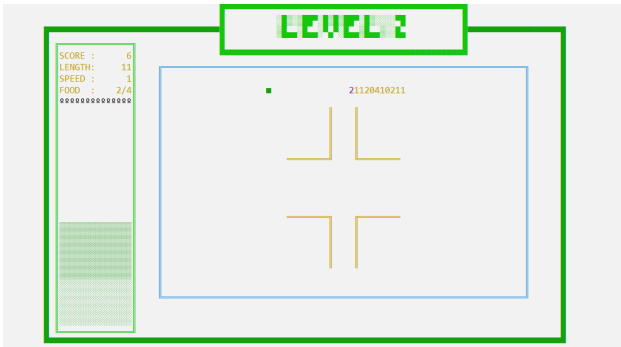
Hình 3.5: Hướng dẫn chơi



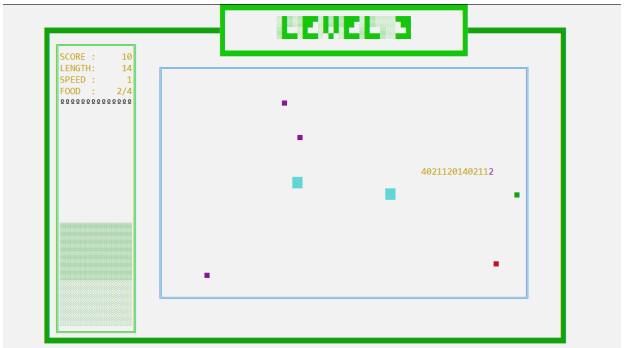
Hình 3.6: Giao diện kết thúc game



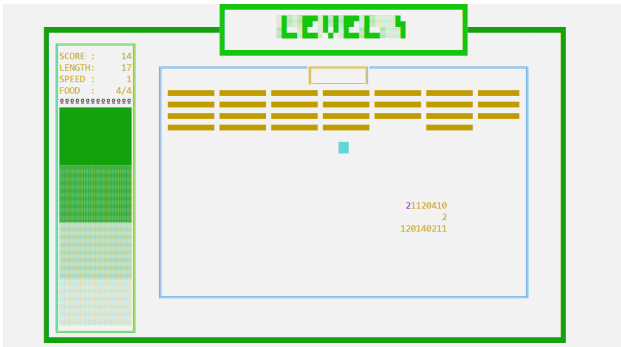
Hình 3.7: Level 1



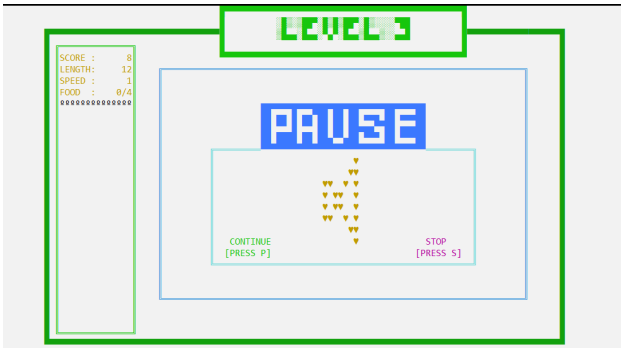
Hình 3.8: Level 2



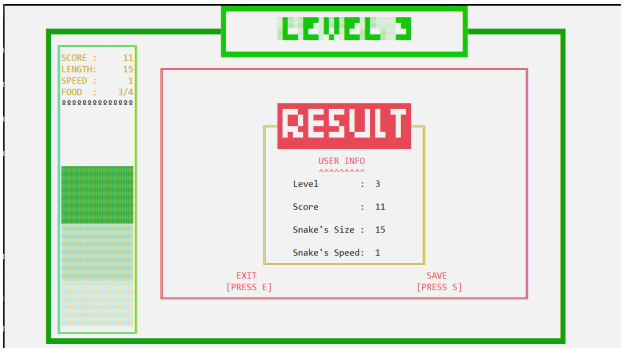
Hình 3.9: Level 3



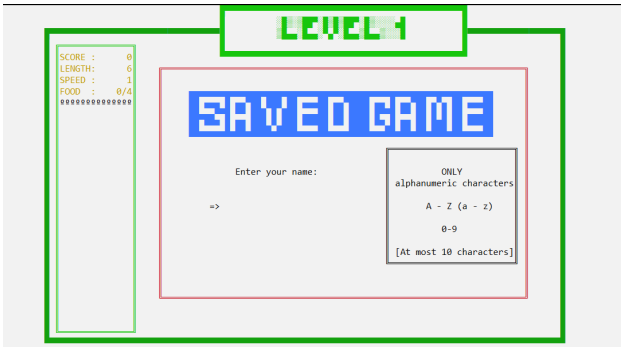
Hình 3.10: Level 4



Hình 3.11: Pause Game



Hình 3.12: Result



Hình 3.13: Save Game

Hình 3.14: Game Saved

Kết luận

Sau khi thực hiện đồ án cuối kì, sản phầm là trò chơi có thể hoạt động được và đảm bảo được các yêu cầu về chức năng đã đề ra. Các chức năng của trò chơi tương đối hoàn chỉnh và thực hiện chính xác. Thuật toán kiểm tra kết quả trò chơi hoạt động chuẩn xác đáp ứng các luật chơi đã đặt ra. Đồ họa bằng màn hình console hiển thị tương đối rõ ràng, dễ nhìn. Các hiệu ứng hình ảnh và âm thanh giúp trò chơi thêm phần sinh động hơn. Do đó, đồ án được đánh giá hoàn thành.

Ngoài ra, qua quá trình làm đồ án, sinh viên được hiểu hơn và biết cách áp dụng kiến thức bao quát của môn học. Đồng thời giúp sinh viên rèn luyện kĩ năng làm việc nhóm, hiểu được quá trình làm game cơ bản bằng phương pháp lập trình hướng thủ tục sử dụng ngôn ngữ C/C++. Hơn nữa đó cũng là cơ hội để sinh viên tìm tòi thêm những kiến thức mới như xử lý giao diện màn hình console, xử lý âm thanh...

Tuy nhiên, trò chơi vẫn còn nhiều điểm hạn chế như: các thuật toán sử dụng chưa tối ưu, nhiều tính năng có thể mở rộng, đồ họa chưa thực sự đẹp mắt... Đây chính là những thiếu sót mà chúng tôi hi vọng có thể khắc phuc trong tương lai.

Tài liệu tham khảo

[1] Hàm giới hạn số kí tự nhập
http://diendan.congdongcviet.com/threads/t75525::lam-the-nao-de-gioi-han-duoc-so-ki
cpp

[2] Tham khảo các mẫu ASCII Art:
https://textfancy.com/multiline-text-art/

[3] In một số kí tự ASCII đặc biệt từ file lên màn hình Console
https://stackoverflow.com/questions/66613855/read-and-print-special-ascii-characters

[4] Sách Kĩ Thuật Lập Trình - NXB Khoa học và Kĩ thuật

[5] Đồ án Game Snake của các khóa trước
https://www.youtube.com/watch?v=OfrRXK3a6pw