TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
Khoa Công nghệ Thông tin
Môn Kỹ thuật Lập trình

ĐỒ ÁN RẮN SĂN MỒI

Nhóm 6

Lớp: 21CLC07

Giảng viên: TS. Trương Toàn Thịnh

Danh sách thành viên

STT	MSSV	Họ và tên
1	21127294	Nguyễn Hi Hữu
2	21127419	Ngô Phước Tài
3	21127693	Huỳnh Đức Thiện
4	21127175	Lê Anh Thư

MỤC LỤC

I. Tổ chức thư mục và quy ước đặt tên trong chương trình

1. Tổ chức thư mục

Thư mục Sound, Data lưu cùng cấp với file mã nguồn đồ án.
Trong đó thư mục Sound chứa các file audio: EatFruit.wav và GameOver.wav; để xử lý âm thanh khi rắn ăn thức ăn (food), kết thúc trò chơi (gameover).
Thư mục Data lưu các file dữ liệu timeRushData.txt, infiniteData.txt, classicData.txt và các file lưu dữ liệu người chơi.

2. Quy ước đặt tên

Đối với tên hằng (constant) thì tất cả các chữ đều viết hoa và các chữ nối nhau bằng dấu gạch chân (snake case).

- Ví dụ: #define MAX_SIZE 40;

Đối với tên struct thì tất cả các chữ đều viết hoa và các chữ viết liền với nhau.

- Ví dụ:
```
struct SNAKELENGTH {
        int length;
        string colour;
};
```

Đối với tên hàm (function) thì viết thường chữ đầu tiên và viết hoa chữ cái đầu tiên của những chữ kế tiếp, các chữ viết liền với nhau (camelCase).

- Ví dụ: void printLogo();

Đối với tên biến (variable) thì tất cả các chữ đều viết thường và các chữ viết liền với nhau.

- Ví dụ: void length = 5;

II. Những định nghĩa, biến được thêm vào

#pragma comment(lib, “winmm.lib”): để có thể thêm nhạc vào code
#pragma warning (disable: 4996): để sử dụng các hàm nhập xuất, xử lý char * của C trong Visual Studio 2019 (mặc định VS 2019 cho rằng cho hàm scanf(), printf(), strcpy(),... là warning)

1. Định nghĩa các thông tin về màn hình

#define PLAY_SCREEN_LENGTH 92: chiều dài màn hình chơi.

#define PLAY_SCREEN_WIDTH 22: chiều dọc màn hình chơi.
#define INFO_BOARD_LENGTH 23: chiều dài bảng thông tin.
#define INFO_BOARD_WIDTH 22: chiều dọc bảng thông tin.
#define LEFT_SIDE_X 26: cạnh trái của màn hình chơi.
#define RIGHT_SIDE_X 118: cạnh phải của màn hình chơi.
#define TOP_SIDE_Y 5: cạnh trên của màn hình chơi.
#define BOTTOM_SIDE_Y 27: cạnh dưới của màn hình chơi.

```
struct POSITION {
        int x;
        int y;
};
```

2. Định nghĩa các thông tin về màu sắc

```
#define BLACK 0
#define DARK_BLUE 1
#define DARK_GREEN 2
#define DARK_CYAN 3
#define DARK_RED 4
#define DARK_PINK 5
#define DARK_YELLOW 6
#define DARK_WHITE 7
#define GREY 8
#define BLUE 9
#define GREEEN 10
#define CYAN 11
#define PINK 12
#define PURPLE 13
#define YELLOW 14
#define WHITE 15
```

3. Định nghĩa các thông tin về game

#define GATE_SPOT 3001: giá trị tại vị trí vào cổng.

```
#define ENTER 13
#define ESC 27
#define UP_ARROW 72
#define DOWN_ARROW 80
#define LEFT_ARROW 75
#define RIGHT_ARROW 77
```

```
struct CLASSICDATA {
        string classicname;
        int score;
        int difficulty;
        int stage;
        SNAKE *snake;
};

struct TIMERUSHDATA {
        string timerushname;
        int score;
        int difficulty;
        int stage;
        SNAKE *snake;
};

struct INFINITEDATA{
        string infinitedata;
        int score;
        SNAKE *snake;
};
```

int gametime = 360: thời gian mặc định lúc bắt đầu chơi trong chế độ Time Rush.
enum eDirection { STOP = 0, LEFT, RIGHT, UP, DOWN, STAY }: gán tên cho các hằng số, dùng để biểu diễn hướng đi hoặc trạng thái của rắn.
const char* stid = "21127175211272942112741921127693”: chuỗi mã số sinh viên dùng để làm thân rắn.
int map[PLAY_SCREEN_WIDTH][PLAY_SCREEN_LENGTH][2] {}: chứa thông tin vật thể của màn chơi

4. Định nghĩa các thông tin về vật thể

#define UPPER_BLOCK 223: nửa khối trên
#define BOTTOM_BLOCK 220: nửa khối dưới
#define BLOCK 219: một khối
#define BUSH_LV1 176: bụi có độ dày thấp
#define BUSH_LV2 177: bụi có độ dày trung bình
#define BUSH_LV3 178: bụi có độ dày cao
#define FOOD 254: tạo hình của thức ăn thông thường

#define TIME_FOOD 235: tạo hình của thức ăn thời gian

```
struct SNAKE {
        POSITION head;
        POSITION *part;
        int size;
};
```

III. Xử lý giao diện chính (main menu)

1. Hàm void fixConsoleWindow(): khóa nút phóng to màn hình console

Dòng	Code
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	}

2. Hàm void gotoXY(): di chuyển tới vị trí trên màn hình console

Dòng	Code
1	void gotoXY(int x, int y) {
2	COORD coord;
3	coord.X = x;
4	coord.Y = y;
5	setConsoleCursorPosition(GetStdHanle(STD_OUTPUT_HANDLE, coord)
6	}

3. Hàm void textColorWithBackground(): viết chữ với màu nền

Dòng	Code
1	void textColorWithBackground(int frontcolor, int backcolor) {
2	WORD wcolor = ((backcolor & 0x0F << 4) + (frontcolor & 0x0F);
3	SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE, wcolor);
4	}

4. Hàm void changeConsoleColor(): đổi màu màn hình console

Dòng	Code
1	void changeConsoleColor(int BackC) {
2	WORD wColor = ((BackC & 0x0F) << 4);
3	Handle hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
4	COORD coord = { 0, 0 };
5	DWORD count;
6	
7	CONSOLE_SCREEN_BUFFER_INFO csbi;
8	
9	SetConsoleTextAttribute(hStdOut, wColor);
10	
11	if (GetConsoleScreenBufferInfo(hStdOut, &csbi) {
12	FillConsoleOutputCharacter(hStdOut, (TCHAR)32, csbi.dwSize.X * csbi.dwSize.Y, coord, &count);
13	FillConsoleOutputAttribute(hStdOut, csbi.wAttributes, csbi.dwSize.X * csbi.dwSize.Y, coord, &count);
14	
15	SetConsoleCursorPosition(hStdOut, coord);
16	}
17	}

5. Hàm void noCursorType(): ẩn con trỏ nhập dữ liệu

Dòng	Code
1	void noCursorType() {
2	CONSOLE_CUROSR_INFO info;
3	info.bVisible = FALSE;
4	info.dwSize = 20;
5	SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &info);
6	}

6. Hàm void drawBoard(): vẽ khung chứa nội dung

Dòng	Code
1	void drawBoard(int x, int y, int length, int width, int color) {
2	textColorWithBackground(color, white);
3	gotoXY(x, y);
4	cout << static_cast<char>(220);
5	for (int i = 1; i < length; i++)
6	cout << static_cast<char>(220);
7	cout << static_cast<char>(220);
8	gotoXY(x, width + y);
9	cout << static_cast<char>(223);
10	for (int i = 0; i < lenth; i++)

11	cout << static_cast<char>(223);
12	
13	for (int i = y + 1; i < width + y; i++) {
14	gotoXY(x, i);
14	cout << static_cast<char>(219);
16	gotoXY(x + length, i);
17	cout << static_cast<char>(219);
18	}
19	}

7. Hàm void drawChoiceBoard(): vẽ ô chọn

Dòng	Code
1	void drawChoiceBox(int x, int y, int length, int width, int color) {
2	gotoXY(x, y);
3	cout << static_cast<char>(201);
4	for (int i = 0; i < length – 1; ++i)
6	cout << static_cast<char>(205);
7	cout << static_cast<char>(187) << endl;
8	for (int i = 1; i <= width – 2; ++i) {
9	gotoXY(x, y + i);
10	cout << static_cast<char>(186);
11	gotoXY(x + length, y + i);
12	cout << static_cast<char>(186);
13	}
14	gotoXY(x, y + width - 1);
14	cout << static_cast<char>(200);
16	for (int i = 0; i < length – 1; ++i)
17	cout << static_cast<char>(205);
19	cout << static_cast<char>(188) << endl;
20	}

8. Hàm void drawBlank(): vẽ khung trống

Dòng	Code
1	void drawBlank(int x, int y, int length, int width) {
2	gotoXY(x, y);
3	for (int i = 0; i < width + 2; i++) {
4	gotoXY(x, y + i);
5	for (int i = 0; i < width + 2; i++) {
	cout << “ “;
6	}

9. Hàm void inputMenuBoxEffect(): tạo hiệu ứng chọn ở menu chính

Dòng	Code
1	void inputMenuBoxEffect(POSITION choice, int color) {
2	if (choice.x == 0 && choice.y == 0) {
3	textColorWithBackground(color, WHITE);
4	drawChoiceBox(27, 14, 17, 3);
5	gotoXY(34, 15);
6	cout << “PLAY”;
7	}
8	if (choice.x == 1 && choice.y == 0) {
9	textColorWithBackground(color, WHITE);
10	drawChoiceBox(75, 14, 17, 3);
11	gotoXY(77, 15);
12	cout << “GAME TURTORIAL”;
13	}
14	if (choice.x == 0 && choice.y == 1) {
15	textColorWithBackground(color, WHITE);
16	drawChoiceBox(27, 22, 17, 3);
17	gotoXY(32, 23);
18	cout << “ABOUT US”;
19	}
20	if (choice.x == 1 && choice.y == 1) {
21	textColorWithBackground(color, WHITE);
22	drawChoiceBox(75, 22, 17, 3);
23	gotoXY(82, 23);
24	cout << “EXIT”;
25	}
26	}

10. Hàm POSITION inputMainMenu(): hàm nhập tín hiệu menu chính

Dòng	Code
1	POSITION inputMainMenu() {
2	POSITION choice {0, 0};
3	POSITION prechoice {0, 0};
4	while (true) {
5	if (_kbhit()) {
6	switch (_getch()) {
7	case ‘a’: case LEFT_ARROW:
8	if (choice.x == 1) {
9	prechoice = choice;
10	choice.x--;
11	inputMenuBoxEffect(prechoice, DARK_YELLOW);

```
12                          inputMenuBoxEffect(choice, DARK_RED);
13                      }
14                      break;
15                  case ‘w’: case UP_ARROW:
16                      prechoice = choice;
17                      if (choice.y == 1) {
18                          prechoice = choice;
19                          choice.y--;
20                          inputMenuBoxEffect(prechoice, DARK_YELLOW);
21                          inputMenuBoxEffect(choice, DARK_RED);
22                      }
23                      break;
24                  case ‘s’: case DOWN_ARROW:
25                      prechoice = choice;
26                      if (choice.y == 0) {
27                          prechoice = choice;
28                          choice.y++;
29                          inputMenuBoxEffect(prechoice, DARK_YELLOW);
30                          inputMenuBoxEffect(choice, DARK_RED);
31                      }
32                      break;
33                  case ‘d’: case RIGHT_ARROW:
34                      prechoice = choice;
35                      if (choice.x == 0) {
36                          prechoice = choice;
37                          choice.x++;
38                          inputMenuBoxEffect(prechoice, DARK_YELLOW);
39                          inputMenuBoxEffect(choice, DARK_RED);
40                      }
41                      break;
42                  case ENTER:
43                      return choice;
44                  }
45              }
46          }
47  }
```

IV. Xử lý giao diện trò chơi (game) và hộp thoại tùy chọn (pause)

1. Hàm void playMenu(): in menu, danh sách các chế độ chơi

Dòng	Code
1	void playMenu() {
2	drawSmallLogo(38, 1);
3	
4	textColorWithBackground(DARK_RED, WHITE);
5	drawChoiceBox(1, 7, 22, 5);

6	gotoXY(9, 9);
7	cout << " CLASSIC";
8	textColorWithBackground(DARK_YELLOW, WHITE);
9	drawChoiceBox(1, 14, 22, 5);
10	gotoXY(8, 16);
11	cout << " TIME RUSH";
12	drawChoiceBox(1, 21, 22, 5);
13	gotoXY(8, 23);
14	cout << " INFINITE";
15	
16	textColorWithBackground(DARK_GREEN, WHITE);
17	drawBoard(26, 5, PLAY_SCREEN_LENGTH, PLAY_SCREEN_WIDTH, PURPLE);
18	}

2. Hàm void inputPlayBoxEffect(): tạo hiệu ứng chọn chế độ chơi

Dòng	Code
1	void inputPlayBoxEffect(POSITION choice, int color) {
2	if (choice.y == 0) {
3	textColorWithBackground(color, WHITE);
4	drawChoiceBox(1, 7, 22, 5);
5	gotoXY(9, 9);
6	cout << "CLASSIC";
8	} else if (choice.y == 1) {
9	textColorWithBackground(color, WHITE);
10	drawChoiceBox(1, 14, 22, 5);
11	gotoXY(8, 16);
12	cout << "TIME RUSH";
13	} else {
14	textColorWithBackground(color, WHITE);
15	drawChoiceBox(1, 21, 22, 5);
16	gotoXY(8, 23);
17	cout << "INFINITE";
18	}
19	}

3. Hàm POSITION inputPlayMenu(): nhập tín hiệu chọn chế độ chơi

Dòng	Code
1	POSITION inputPlayMenu() {
2	POSITION choice {0, 0};
3	POSITION prechoice {0, 0};
4	while (true) {
5	if (_kbhit()) {
6	switch (_getch()) {

7	case ‘w’: case UP_ARROW:
8	if (choice.y != 0) {
9	prechoice = choice;
10	choice.y--;
11	inputMenuBoxEffect(prechoice, DARK_YELLOW);
12	inputMenuBoxEffect(choice, DARK_RED);
13	}
14	break;
15	case ‘s’: case DOWN_ARROW:
16	prechoice = choice;
17	if (choice.y != 2) {
18	prechoice = choice;
19	choice.y++;
20	inputMenuBoxEffect(prechoice, DARK_YELLOW);
21	inputMenuBoxEffect(choice, DARK_RED);
22	}
23	break;
24	case ESC:
25	choice.x = 1;
26	return choice;
27	case ENTER:
28	return choice;
29	}
30	}
31	}
32	}

4. Hàm int inputTimeChoice(): nhập tín hiệu chọn màn chơi

Dòng	Code
1	void inputTimeChoice() {
2	drawBlank(57, 9, 30 , 9);
3	drawBoard(57, 9, 30, 9, PINK);
4	gotoXY(67, 11);
5	cout << “[1] Map 1”;
6	gotoXY(67, 12);
7	cout << “[2] Map 2”;
8	gotoXY(67, 13);
9	cout << “[3] Map 3”;
10	gotoXY(67, 14);
11	cout << “[4] Map 4”;
12	gotoXY(67, 15);
13	cout << “[5] Map 5”;
14	drawBoard(57, 18, 30, 4, PINK);
15	gotoXY(57, 18);
16	for (int i = 0; i < 31; i++) {
17	cout << char(BLOCK);

```
18      }
19      gotoXY(66, 20);
20      cout << “Chose a map: “;
21      int choice;
22      cin >> choice;
23      choice = choice % 5;
24      if (choice == 0)
25          choice = 5;
26      return choice;
27  }
```

5. Hàm void infoBoard(): in bảng thông tin

Dòng	Code
1	void infoBoard(int x, int y) {
2	drawBlank(x, y, INFO_BOARD_LENGTH, INFO_BOARD_WIDTH);
3	gotoXY(x, y);
4	textColorWithBackground(PURPLE, WHITE);
5	for (int i = 0; i < INFO_BOARD_LENGTH + 2; ++i) {
6	textColorWithBackground(PURPLE, WHITE);
7	cout << static_cast<char>(220);
8	Sleep(10);
9	}
10	for (int i = 1; i < INFO_BOARD_WIDTH; ++i) {
11	gotoXY(x + INFO_BOARD_LENGTH + 1; y + i);
12	textColorWithBackground(PURPLE, WHITE);
13	cout << static_cast<char>(219);
14	Sleep(10);
15	}
16	for (int i = x + INFO_BOARD_LENGTH; i >= x - 1; --i) {
17	gotoXY(x + i; y + INFO_BOARD_WIDTH);
18	textColorWithBackground(PURPLE, WHITE);
19	cout << static_cast<char>(223);
20	Sleep(10);
21	}
22	for (int i = y + INFO_BOARD_WIDTH - 1; i > y; --i) {
23	gotoXY(x, i);
24	textColorWithBackground(PURPLE, WHITE);
25	cout << static_cast<char>(219);
26	Sleep(10);
27	}
28	}

6. Hàm void infoSet(): tạo thông tin cho bảng thông tin

Dòng	Code
1	void infoSet(int type) {
2	textColorWithBackground(DARK_YELLOW, WHITE);
3	if (type == 1) {
4	gotoXY(3, 7);
5	cout << “Stage: “;
6	}
7	if (type == 3) {
8	gotoXY(3, 7);
9	cout << “Speed: “;
10	}
11	
12	gotoXY(3, 9);
13	cout << “Score: “;
14	
15	gotoXY(3, 11);
16	cout << “(p): Pause”;
17	}

7. Hàm void pauseGameBoard(): in bảng tạm dừng

Dòng	Code
1	void pauseGameBoard(int x, int y) {
2	drawBlank(x, y, 22, 7);
3	textColorWithBackground(PINK, WHITE);
4	for (int i = 0; i < 8; ++i) {
5	gotoXY(x + 22, y + i);
6	cout << char(219);
7	}
8	for (int i = y + 8; i > y; --i) {
9	gotoXY(x + 1, i);
10	cout << char(219);
11	}
12	gotoXY(x + 1, y);
13	for (int i = 0; i < 22; ++i) {
14	cout << char(220);
15	}
16	for (int i = x + 22; i >= x + 1; --i) {
17	gotoXY(i, y + 8);
18	cout << char(223);
19	}
20	gotoXY(x + 4, y + 2);
21	cout << “(c): continue”;
22	gotoXY(x + 4, y + 4);
23	cout << “(s): save game”;

24	gotoXY(x + 4, y + 6);
25	cout << “(x): exit”;
26	}

8. Hàm void inputSubChoiceEffect(): tạo hiệu ứng khi chọn giữa độ khó và chơi lại màn đã lưu

Dòng	Code
1	void inputSubChoiceEffect(POSITION choice, int color) {
2	if (choice.y == 0) {
3	textColorWithBackground(color, WHITE);
4	drawChoiceBox(1, 10, 22, 5);
5	cout << “DIFFICULTY”;
6	} else {
7	textColorWithBackground(color, WHITE);
8	drawChoiceBox(1, 19, 22, 5);
9	gotoXY(8, 21);
10	cout << “LOAD GAME”;
11	}
12	}

9. Hàm POSITION inputSubChoiceMenu(): nhận tín hiệu lựa chọn giữa độ khó và chơi lại màn đã lưu

Dòng	Code
1	POSITION inputPlayMenu() {
2	POSITION choice {0, 0};
3	POSITION prechoice {0, 0};
4	while (true) {
5	if (_kbhit()) {
6	switch (_getch()) {
7	case ‘w’: case UP_ARROW:
8	if (choice.y != 0) {
9	prechoice = choice;
10	choice.y--;
11	inputSubChoiceEffect(prechoice, DARK_YELLOW);
12	inputSubChoiceEffect(choice, DARK_RED);
13	}
14	break;
15	case ‘s’: case DOWN_ARROW:
16	if (choice.y != 1) {
17	prechoice = choice;
18	choice.y++;
19	inputSubChoiceEffect(prechoice, DARK_YELLOW);
20	inputSubChoiceEffect(choice, DARK_RED);
21	}

22	break;
23	case ESC:
24	choice.x = 1;
25	return choice;
26	case ENTER:
27	return choice;
28	}
29	}
30	}
31	}

10. Hàm POSITION subChoiceMenu(): in lựa chọn giữa chọn độ khó và chơi lại màn đã lưu

Dòng	Code
1	POSITION subChoiceMenu() {
2	drawBlank(1, 5, 21, 20);
3	textColorWithBackground(DARK_RED, WHITE);
4	drawChoiceBox(1, 10, 22, 5);
5	gotoXY(7, 12);
6	cout << “DIFFICULTY”;
8	
9	textColorWithBackground(DARK_YELLOW, WHITE);
10	drawChoiceBox(1, 19, 22, 5);
11	gotoXY(8, 21);
12	cout << “LOAD GAME”;
13	
14	return inputSubChoiceMenu();
15	}

11. Hàm void inputInfiniteSubChoiceEffect(): hàm tạo hiệu ứng khi chọn trong chế độ chơi Infinite

Dòng	Code
1	void inputInfiniteSubChoiceEffect(POSITION choice, int color) {
2	if (choice.y == 0) {
3	textColorWithBackground(color, WHITE);
4	drawChoiceBox(1, 10, 22, 5);
	gotoXY(8, 12);
5	cout << “NEW GAME”;
6	} else {
7	textColorWithBackground(color, WHITE);
8	drawChoiceBox(1, 19, 22, 5);
9	gotoXY(8, 21);
10	cout << “LOAD GAME”;

11	}
12	}

12. Hàm POSITION inputInfiniteSubChoiceEffect(): hàm nhập tín hiệu chọn trong chế độ Infinite

Dòng	Code
1	POSITION inputPlayMenu() {
2	POSITION choice {0, 0};
3	POSITION prechoice {0, 0};
4	while (true) {
5	if (_kbhit()) {
6	switch (_getch()) {
7	case ‘w’: case UP_ARROW:
8	if (choice.y != 0) {
9	prechoice = choice;
10	choice.y--;
11	inputInfiniteSubChoiceEffect(prechoice, DARK_YELLOW);
12	inputInfiniteSubChoiceEffect(choice, DARK_RED);
13	}
14	break;
15	case ‘s’: case DOWN_ARROW:
16	if (choice.y != 1) {
17	prechoice = choice;
18	choice.y++;
19	inputInfiniteSubChoiceEffect(prechoice, DARK_YELLOW);
20	inputInfiniteSubChoiceEffect(choice, DARK_RED);
21	}
22	break;
23	case ESC:
24	choice.x = 1;
25	return choice;
26	case ENTER:
27	return choice;
28	}
29	}
30	}
31	}

13. Hàm POSITION infiniteSubChoiceMenu(): hàm chọn các nội dung trong chế độ chơi Infinite

Dòng	Code
1	POSITION infinitesubChoiceMenu() {
2	drawBlank(1, 5, 21, 20);
3	textColorWithBackground(DARK_RED, WHITE);
4	drawChoiceBox(1, 10, 22, 5);
5	gotoXY(8, 12);
6	cout << “NEW GAME”;
8	
9	textColorWithBackground(DARK_YELLOW, WHITE);
10	drawChoiceBox(1, 19, 22, 5);
11	gotoXY(8, 21);
12	cout << “LOAD GAME”;
13	
14	return inputInfiniteSubChoiceMenu();
15	}

14. Hàm void hoverDifficultyChoice(): hàm tạo hiệu ứng khi chọn độ khó

Dòng	Code
1	void hoverDifficultyChoice(int choice, int prechoice) {
2	textColorWithBackground(PURPLE, WHITE);
3	switch (choice) {
4	case 0:
5	gotoXY(40, 11);
6	cout << “>> Easy ”;
7	break;
8	case 1:
9	gotoXY(40, 16);
10	cout << “>> Normal ”;
11	break;
12	case 2:
13	gotoXY(40, 21);
14	cout << “>> Hard ”;
15	break;
16	}
17	
18	textColorWithBackground(DARK_PINK, WHITE);
19	switch(prechoice) {
20	case 0:
21	gotoXY(40, 11);
22	cout << “>> Easy ”;
23	break;

24	case 1:
25	gotoXY(40, 16);
26	cout << “>> Normal ”;
27	break;
28	case 2:
29	gotoXY(40, 21);
30	cout << “>> Hard ”;
31	break;
32	}
33	}

15. Hàm int inputDifficultyChoice(): hàm nhận tín hiệu chọn độ khó

Dòng	Code
1	int inputDifficultyChoice(int choice) {
2	int prechoice = choice + 1;
3	while (true) {
4	if (_kbhit()) {
5	switch (_getch()) {
6	case ‘s’: case DOWN_ARROW:
7	if (choice != 2)
8	prechoice = choice;
9	choice++;
10	}
11	break;
12	case ‘w’: case UP_ARROW:
13	if (choice != 0)
14	prechoice = choice;
15	choice--;
16	}
17	break;
18	case ESC:
19	return -1;
20	case ENTER:
21	retrun choice;
22	}
23	}
24	hoverDifficultyChoice(choice, prechoice);
25	}
26	}

16. Hàm int choseDifficulty(): hàm chọn độ khó cho màn chơi

Dòng	Code
1	int choseDifficulty() {

2	int choice = 0;
3	textColorWithBackground(PURPLE, WHITE);
4	gotoXY(40, 11);
5	cout << “>> Easy”;
6	textColorWithBackground(DARK_PINK, WHITE);
7	gotoXY(40, 16);
8	cout << “Normal”;
9	gotoXY(40, 21);
10	cout << “Hard”;
11	return inputDifficultyChoice(choice);
12	}

17. Hàm void hoverClassicPlayerChoice(): hàm hiệu ứng khi chọn màn chơi đã lưu trong chế độ Classic

Dòng	Code
1	void hoverClassicPlayerChoice(int choice, int prechoice, CLASSICDATA *data, int playercount) {
2	if (choice == prechoice) return;
3	textColorWithBackground(PURPLE, WHITE);
4	gotoXY(28, 7 + choice * 2);
5	cout << “>> “ << data[choice].classicname;
6	gotoXY(68, 7 + choice * 2);
7	cout << data[choice].score;
8	gotoXY(102, 7 + choice * 2);
9	
10	if (data[choice].difficulty == 0) {
11	cout << “Easy”;
12	} else if (data[choice].difficulty == 1) {
13	cout << “Normal”;
14	} else {
15	cout << “Hard”;
16	}
17	
18	textColorWithBackground(DARK_PINK, WHITE);
19	gotoXY(28, 7 + prechoice * 2);
20	cout << “>> “ << data[prechoice].classicname << “ “;
21	gotoXY(68, 7 + prechoice * 2);
22	cout << data[prechoice].score;
23	gotoXY(102, 7 + prechoice * 2);
24	
25	if (data[prechoice].difficulty == 0) {
26	cout << “Easy”;
27	} else if (data[prechoice].difficulty == 1) {
28	cout << “Normal”;
29	} else {
30	cout << “Hard”;

31	}
32	}

18. Hàm int choseClassicPlayer(): hàm chọn người chơi đã lưu trong chế độ Classic

Dòng	Code
1	int choseClassicPlayer(int choice, CLASSICDATA *data, int playercount) {
2	int prechoice = choice;
3	int limit = 0;
4	if (playercount <= 10) limit = playercount - 1;
5	else limit = 9;
6	while (true) {
7	if (_kbhit()) {
8	switch (_getch()) {
9	case ‘s’: case DOWN_ARROW:
10	if (choice < limit)
11	prechoice = choice;
12	choice++;
13	}
14	break;
15	case ‘w’: case UP_ARROW:
16	if (choice > 0)
17	prechoice = choice;
18	choice--;
19	}
20	break;
21	case ESC:
22	return -1;
23	case ENTER:
24	retrun choice;
25	}
26	}
27	hoverClassicPlayerChoice(choice, prechoice, data, playercount);
28	}
29	}

19. Hàm loadSaveClassicPlayer(): hàm in danh sách các người chơi đã lưu trong chế độ Classic

Dòng	Code
1	int loadSaveClassicPlayer(CLASSICDATA *data, int playercount)
2	if (playercount <= 0) {
3	gotoXY(28, 7);
4	cout << “Empty Data!”;
5	return -1;

```
6     } else {
7         drawBoard(26, 5, PLAY_SCREEN_LENGTH,
PLAY_SCREEN_WIDTH, YELLOW);
8         int choice = 0;
9         textColorWithBackground(PURPLE, WHITE);
10        gotoXY(36, 6);
11        cout << “NAME”;
12        gotoXY(67, 6);
13        cout << “SCORE”;
14        gotoXY(100, 6);
15        cout << “DIFFICULTY”;
16
17        textColorWithBackground(DARK_PINK, WHITE);
18        if (playercount <= 10) {
19            for (int i = 0; i < playercount; i++) {
20                gotoXY(28, 7 + 2 * i);
21                cout << data[i].classicname;
22                gotoXY(68, 7 + 2 * i);
23                cout < data[i].score;
24                gotoXY(102, 7 + 2 * i);
25                if (data[i].difficulty == 0) {
26                    cout << “Easy”;
27                }
28                else if (data[i].difficulty == 1) {
29                    cout << “Normal”;
30                }
31                else {
32                    cout << “Hard”;
33                }
34            } else {
35                for (int i = 0; i < 10; i++) {
36                    gotoXY(28, 7 + 2 * i);
37                    cout << data[i].classicname;
38                    gotoXY(68, 7 + 2 * i);
39                    cout < data[i].score;
40                    gotoXY(102, 7 + 2 * i);
41                    if (data[i].difficulty == 0) {
42                        cout << “Easy”;
43                    }
44                    else if (data[i].difficulty == 1) {
45                        cout << “Normal”;
46                    }
47                    else {
48                        cout << “Hard”;
49                    }
50                }
51            }
52
53        textColorWithBackground(PURPLE, WHITE);
```

54	gotoXY(27, 7);
55	for (int i = 0; i < PLAY_SCREEN_LENGTH - 4; i++) {
56	cout << " ";
57	}
58	gotoXY(28, 7);
59	cout << ">> " << data[choice].classicname;
60	gotoXY(68, 7);
61	cout <<< data[choice].score;
62	gotoXY(102, 7);
63	if (data[choice].difficulty == 0) {
64	cout << "Easy";
65	}
66	else if (data[choice].difficulty == 1) {
67	cout << "Normal";
68	}
69	else {
70	cout << "Hard";
71	}
72	return choseClassicPlayer(choice, data, playercount);
73	}
74	}

20. Hàm void hoverTimeRushPlayerChoice(): hàm hiệu ứng khi chọn màn chơi đã lưu trong chế độ Time Rush

Dòng	Code
1	void hoverTimeRushPlayerChoice(int choice, int prechoice, TIMERUSH *data, int playercount) {
2	if (choice == prechoice) return;
3	textColorWithBackground(PURPLE, WHITE);
4	gotoXY(28, 7 + choice * 2);
5	cout << ">> " << data[choice].timerushname;
6	gotoXY(68, 7 + choice * 2);
7	cout << data[choice].score;
8	gotoXY(102, 7 + choice * 2);
9	
10	if (data[choice].difficulty == 0) {
11	cout << "Easy";
12	} else if (data[choice].difficulty == 1) {
13	cout << "Normal";
14	} else {
15	cout << "Hard";
16	}
17	
18	textColorWithBackground(DARK_PINK, WHITE);
19	gotoXY(28, 7 + prechoice * 2);
20	cout << ">> " << data[prechoice].timerushname<< " ";

21	gotoXY(68, 7 + prechoice * 2);
22	cout << data[prechoice].score;
23	gotoXY(102, 7 + prechoice * 2);
24	
25	if (data[prechoice].difficulty == 0) {
26	cout << “Easy”;
27	} else if (data[prechoice].difficulty == 1) {
28	cout << “Normal”;
29	} else {
30	cout << “Hard”;
31	}
32	}

21. Hàm int choseTimeRushPlayer(): hàm chọn người chơi đã lưu trong chế độ Time Rush

Dòng	Code
1	int choseTimeRushPlayer(int choice, TIMERUSHDATA *data, int playercount) {
2	int prechoice = choice;
3	int limit = 0;
4	if (playercount <= 10) limit = playercount - 1;
5	else limit = 9;
6	while (true) {
7	if (_kbhit()) {
8	switch (_getch()) {
9	case ‘s’: case DOWN_ARROW:
10	if (choice < limit)
11	prechoice = choice;
12	choice++;
13	}
14	break;
15	case ‘w’: case UP_ARROW:
16	if (choice > 0)
17	prechoice = choice;
18	choice--;
19	}
20	break;
21	case ESC:
22	return -1;
23	case ENTER:
24	retrun choice;
25	}
26	}
27	hoverTimeRushPlayerChoice(choice, prechoice, data, playercount);
28	}
29	}

22. Hàm int loadSaveTimeRushPlayer(): hàm in danh sách các người chơi đã lưu trong chế độ Time Rush

Dòng	Code
1	int loadSaveTimeRushPlayer(TIMERUSHDATA *data, int playercount)
2	if (playercount <= 0) {
3	gotoXY(28, 7);
4	cout << “Empty Data!”;
5	return -1;
6	} else {
7	drawBoard(26, 5, PLAY_SCREEN_LENGTH, PLAY_SCREEN_WIDTH, YELLOW);
8	int choice = 0;
9	textColorWithBackground(PURPLE, WHITE);
10	gotoXY(36, 6);
11	cout << “NAME”;
12	gotoXY(67, 6);
13	cout << “SCORE”;
14	gotoXY(100, 6);
15	cout << “DIFFICULTY”;
16	
17	textColorWithBackground(DARK_PINK, WHITE);
18	if (playercount <= 10) {
19	for (int i = 0; i < playercount; i++) {
20	gotoXY(28, 7 + 2 * i);
21	cout << data[i].timerushname;
22	gotoXY(68, 7 + 2 * i);
23	cout < data[i].score;
24	gotoXY(102, 7 + 2 * i);
25	if (data[i].difficulty == 0) {
26	cout << “Easy”;
27	}
28	else if (data[i].difficulty == 1) {
29	cout << “Normal”;
30	}
31	else {
32	cout << “Hard”;
33	}
34	} else {
35	for (int i = 0; i < 10; i++) {
36	gotoXY(28, 7 + 2 * i);
37	cout << data[i].timerushname;
38	gotoXY(68, 7 + 2 * i);
39	cout < data[i].score;
40	gotoXY(102, 7 + 2 * i);
41	if (data[i].difficulty == 0) {
42	cout << “Easy”;
43	}

44	else if (data[i].difficulty == 1) {
45	cout << “Normal”;
46	}
47	else {
48	cout << “Hard”;
49	}
50	}
51	}
52	
53	textColorWithBackground(PURPLE, WHITE);
54	gotoXY(27, 7);
55	for (int i = 0; i < PLAY_SCREEN_LENGTH - 4; i++) {
56	cout << “ “;
57	}
58	gotoXY(28, 7);
59	cout << “>> “ << data[choice].timerushname;
60	gotoXY(68, 7);
61	cout <<< data[choice].score;
62	gotoXY(102, 7);
63	if (data[choice].difficulty == 0) {
64	cout << “Easy”;
65	}
66	else if (data[choice].difficulty == 1) {
67	cout << “Normal”;
68	}
69	else {
70	cout << “Hard”;
71	}
72	return choseTimeRushPlayer(choice, data, playercount);
73	}
74	}

23. Hàm void hoverInfinitePlayerChoice(): hàm hiệu ứng khi chọn màn chơi đã lưu trong chế độ Infinite

Dòng	Code
1	void hoverInfinitePlayerChoice(int choice, int prechoice, INFINITEDATA *data, int playercount) {
2	if (choice == prechoice) return;
3	textColorWithBackground(PURPLE, WHITE);
4	gotoXY(28, 7 + choice * 2);
5	cout << “>> “ << data[choice].infinitename;
6	gotoXY(68, 7 + choice * 2);
7	cout << data[choice].score;
8	
9	textColorWithBackground(DARK_PINK, WHITE);
10	gotoXY(28, 7 + prechoice * 2);

11	cout << “>> “ << data[prechoice].infinitename<< “ “;
12	gotoXY(68, 7 + prechoice * 2);
13	cout << data[prechoice].score;
14	}

24. Hàm int choseInfinitePlayer(): hàm chọn người chơi đã lưu trong chế độ Infinite

Dòng	Code
1	int choseTimeRushPlayer(int choice, INFINITEDATA *data, int playercount) {
2	int prechoice = choice;
3	int limit = 0;
4	if (playercount <= 10) limit = playercount - 1;
5	else limit = 9;
6	while (true) {
7	if (_kbhit()) {
8	switch (_getch()) {
9	case ‘s’: case DOWN_ARROW:
10	if (choice < limit)
11	prechoice = choice;
12	choice++;
13	}
14	break;
15	case ‘w’: case UP_ARROW:
16	if (choice > 0)
17	prechoice = choice;
18	choice--;
19	}
20	break;
21	case ESC:
22	return -1;
23	case ENTER:
24	retrun choice;
25	}
26	}
27	hoverInfinitePlayerChoice(choice, prechoice, data, playercount);
28	}
29	}

25. Hàm int loadSaveInfinitePlayer(): hàm in danh sách các người chơi đã lưu trong chế độ Infinite

Dòng	Code
1	int loadSaveInfinitePlayer(INFINITEDATA *data, int playercount)
2	if (playercount <= 0) {
3	gotoXY(28, 7);
4	cout << “Empty Data!”;
5	return -1;
6	} else {
7	drawBoard(26, 5, PLAY_SCREEN_LENGTH, PLAY_SCREEN_WIDTH, YELLOW);
8	int choice = 0;
9	textColorWithBackground(PURPLE, WHITE);
10	gotoXY(36, 6);
11	cout << “NAME”;
12	gotoXY(67, 6);
13	cout << “SCORE”;
14	
15	textColorWithBackground(DARK_PINK, WHITE);
16	if (playercount <= 10) {
17	for (int i = 0; i < playercount; i++) {
18	gotoXY(28, 7 + 2 * i);
19	cout << data[i].infinitename;
20	gotoXY(68, 7 + 2 * i);
21	}
22	} else {
23	for (int i = 0; i < 10; i++) {
24	gotoXY(28, 7 + 2 * i);
25	cout << data[i].infinitename;
26	gotoXY(68, 7 + 2 * i);
27	cout < data[i].score;
28	}
29	}
30	
31	textColorWithBackground(PURPLE, WHITE);
32	gotoXY(27, 7);
33	for (int i = 0; i < PLAY_SCREEN_LENGTH - 4; i++) {
34	cout << “ “;
35	}
36	gotoXY(28, 7);
37	cout << “>> “ << data[choice].infinitename;
38	gotoXY(68, 7);
39	cout <<< data[choice].score;
40	
41	return choseTimeRushPlayer(choice, data, playercount);
42	}
43	}

26. Hàm string inputName(): nhập tên từ người chơi

Dòng	Code
1	string inputName() {
2	drawBoard(57, 14, 30, 4, DARK_GREEN);
3	gotoXY(60, 16);
4	cout << “Name: “;
5	textColorWithBackground(BLACK, WHITE);
6	string name;
7	getline(cin, name);
8	return name;
9	}

V. Xử lý rắn (snake), thức ăn (food) và bản đồ (map)

1. Hàm void gameSetup(): khởi tạo các giá trị ban đầu trước khi chơi

Dòng	Code
1	void gameSetup() {
2	dir = STAY;
3	srand(time(NULL));
4	score = 0;
5	gameover = false;
6	save = false;
7	foodnum = 1;
8	stage = 1;
9	charlock = STOP;
10	snake->part = new POSITION[100];
11	snake->head = {{RIGHT_SIDE_X + LEFT_SIDE_X) / 2, (BOTTOM_SIDE_Y + TOP_SIDE_Y) / 2};
12	snake->part[0].y = snake->head.y;
13	snake->part[0].x = snake->head.x + 1;
14	snake->size = 1;
15	}

2. Hàm void getDir(): nhận tín hiệu di chuyển cho rắn từ bàn phím

Dòng	Code
1	void getDir() {
2	if (_kbhit()) {
3	switch(_getch()) {
4	case ‘w’: case UP_ARROW:
5	if (charlock != UP) {
6	dir = UP;
7	charlock = DOWN;

```
8                   }
9                   break;
10              case ‘a’: case LEFT_ARROW:
11                  if (charlock != LEFT) {
12                      dir = LEFT;
13                      charlock = RIGHT;
14                  }
15                  break;
16              case ‘s’: case DOWN_ARROW:
17                  if (charlock != DOWN) {
18                      dir = DOWN;
19                      charlock = UP;
20                  }
21                  break;
22              case ‘d’: case RIGHT_ARROW:
23                  if (charlock != RIGHT) {
24                      dir = RIGHT;
25                      charlock = LEFT;
26                  }
27                  break;
28               case ‘p’:
29                  temp = dir;
30                  dir = STOP;
31                  break;
32               default:
33                  break;
34          }
35      }
36  }
```

3. Hàm void moving(): định hướng di chuyển cho rắn

Dòng	Code
1	void moving(int type) {
2	switch(dir) {
3	case LEFT:
4	snake->head.x--;
5	break;
6	case DOWN:
7	snake->head.y++;
8	break;
9	case UP:
10	snake->head.y--;
11	break;
12	case RIGHT:
13	snake->head.x++;
14	break;
15	case STOP:

16	pauseGameBoard(60, 13);
17	pauseGameInput(60, 13, type)’
18	break;
19	default:
20	break;
21	}
22	}

4. Hàm void swapSide(): hàm chuyển đổi bên cho chế độ Infinite

Dòng	Code
1	void swapSide() {
2	if (snake->head.x >= RIGHT_SIDE_X) {
3	snake->head.x = LEFT_SIDE_X + 1;
4	} else if (snake->head.x <= LEFT_SIDE_X) {
5	snake->head.x = RIGHT_SIDE_X - 1;
6	}
7	
8	if (snake->head.y >= BOTTOM_SIDE_Y) {
9	snake->head.y = TOP_SIDE_Y + 1;
10	} else if (snake->head.y <= TOP_SIDE_Y) {
11	snake->head.y = BOTTOM_SIDE_Y - 1;
12	}
13	}

5. Hàm bool eatFood(): Kiểm tra rắn đã ăn thức ăn chưa

Dòng	Code
1	bool eatFood() {
2	if (snake->head.x == food.x && snake->head.y == food.y) return true;
3	return false;
4	}

6. Hàm renderSnake(): xử lý vị trí đầu rắn và thân rắn

Dòng	Code
1	void renderSnake(POSITION head) {
2	POSITION prevpart, prev2part;
3	prevpart = snake->part[0];
4	snake->part[0] = head;
5	for (int idx = 1; idx < snake->size; idx++) {
6	prev2part = snake->part[idx];
7	snake->part[idx] = prevpart;
8	prevpart = prev2part;
9	}

10	}

7. Hàm void printFood(): in thức ăn thường

Dòng	Code
1	void printFood(POSITION food) {
2	textColorWithBackground(DARK_YELLOW, WHITE);
3	gotoXY(food.x, food.y);
4	if (!isFoodInBush(food)) {
5	cout << char(FOOD);
6	} else {
7	cout << char.(map[food.y - TOP_SIDE_Y][food.x - LEFT_SIDE_X][0]);
8	}
9	}

8. Hàm void clearFood(): xóa thức ăn cũ

Dòng	Code
1	void clearFood(POSITION food) {
2	gotoXY(food.x, food.y);
3	if (map[food.y - TOP_SIDE_Y][food.x - LEFT_SIDE_X][0] <= BUSH_LV3 && map[food.y - TOP_SIDE_Y][food.x - LEFT_SIDE_X][0] >= BUSH_LV1) {
4	textColorWithBackground(DARK_GREEN, WHITE);
5	cout << char(map[food.y - TOP_SIDE_Y][food.x - LEFT_SIDE_X][0]);
6	} else {
7	textColorWithBackground(WHITE, WHITE);
8	cout << " ";
9	}
10	}

9. Hàm void printTimeFood(): in thức ăn thời gian

Dòng	Code
1	void printTimeFood(POSITION timefood) {
2	textColorWithBackground(PURPLE, WHITE);
3	gotoXY(timefood.x, timefood.y);
4	if (!isFoodInBush(timefood)) {
5	cout << char(TIME_FOOD);
6	} else {
7	cout << char.(map[food.y - TOP_SIDE_Y][food.x - LEFT_SIDE_X][0]);
8	}
9	}

10. Hàm void printGate(): in cổng qua màn

Dòng	Code
1	void printGate(int x, int y, int type) {
2	switch(type) {
3	case 0:
4	textColorWithBackground(DARK_YELLOW, WHITE);
5	gotoXY(x - 1, y -1);
6	for (int i = 0; i < 4; i++) {
7	cout << char(BOTTOM_BLOCK):
8	}
9	textColorWithBackground(DARK_YELLOW, BLACK);
10	gotoXY(x - 1, y);
11	cout << char(BLOCK) << " " << char(BLOCK);
12	break;
13	case 1:
14	textColorWithBackground(DARK_YELLOW, WHITE);
15	gotoXY(x - 1, y -1);
16	cout << char(BOTTOM_BLOCK) << char(BOTTOM_BLOCK);
17	textColorWithBackground(DARK_YELLOW, BLACK);
18	gotoXY(x - 1, y);
19	cout << char(BLOCK) << ' ';
20	textColorWithBackground(DARK_YELLOW, WHITE);
21	gotoXY(x - 1, y + 1);
22	cout << char(UPPER_BLOCK) << char(UPPER_BLOCK);
23	break;
24	}
25	}

11. Hàm void castMapBase(): khởi tạo lại giá trị cơ bản cho màn chơi

Dòng	Code
1	void castMapBase() {
2	for (int i = 0; i < PLAY_SCREEN_WIDTH, i++) {
3	for (int j = 0l j < PLAY_SCREEN_LENGTH; j++) {
4	map[i][j][0] = 0;
5	map[i][j][1] = WHITE;
6	}
7	}
8	}

12. Hàm void printMap(): in màn chơi

Dòng	Code
1	void printMap() {
2	for (int i = 0; i < PLAY_SCREEN_WIDTH, i++) {
3	for (int j = 0l j < PLAY_SCREEN_LENGTH; j++) {
4	if (map[i][j][0] != 15) {
5	textColorWithBackground(map[i][j][1], WHITE);
6	gotoXY(j + LEFT_SIDE_X, i + TOP_SIDE_Y);
7	cout << char(map[i][j][0]);
8	}
9	}
10	}
11	}

13. Hàm void drawDesGate(): in cổng đích

Dòng	Code
1	void drawDesGate(int type, int x, int y) {
2	textColorWithBackground(DARK_YELLOW, WHITE)’
3	if (type == 1) {
4	gotoXY(x - 1, y);
5	cout << char(214) << ‘0’ << char(183);
6	gotoXY(x - 1, y + 1);
7	cout << char(186) << ‘ ‘ << char(186);
8	gotoXY(x - 1, y + 1);
9	cout << char(189) << ‘ ‘ << char(211);
10	} else {
11	gotoXY(x - 1, y);
12	cout << char(213) << char(205) << char(190);
13	gotoXY(x, y);
14	cout << ‘0‘;
15	gotoXY(x, y + 1);
16	cout << char(212) << char(205) << char(184);
17	}
18	}

14. Hàm void desGate(): tạo vùng vị trí cổng đích

Dòng	Code
1	void desGate(int stage) {
2	switch(stage) {
3	case 1: case 2:
4	drawDesGate(1, LEFT_SIDE_X + 2, TOP_SIDE_Y + 1);
5	snake->head = {LEFT_SIDE_X + 1, TOP_SIDE_Y + 2};
6	for (int i = 0; i < snake->size; i++) {

7	snake->part[i] = snake->head:
8	}
9	dir = DOWN;
10	charlock = UP;
11	break;
12	case 3:
13	drawDesGate(2, LEFT_SIDE_X + 1, TOP_SIDE_Y + 4);
14	snake->head = {LEFT_SIDE_X + 2, TOP_SIDE_Y + 4};
15	for (int i = 0; i < snake->size; i++) {
16	snake->part[i] = snake->head:
17	}
18	dir = RIGHT;
19	charlock = LEFT;
20	break;
21	case 4:
22	drawDesGate(1, RIGHT_SIDE_X - 5, TOP_SIDE_Y + 1);
23	snake->head = {RIGHT_SIDE_X - 5, TOP_SIDE_Y + 2};
24	for (int i = 0; i < snake->size; i++) {
25	snake->part[i] = snake->head:
26	}
27	dir = DOWN;
28	charlock = UP;
29	break;
30	case 5:
31	drawDesGate(2, LEFT_SIDE_X + 1, TOP_SIDE_Y + 14);
32	snake->head = {LEFT_SIDE_X + 3, TOP_SIDE_Y + 14};
33	for (int i = 0; i < snake->size; i++) {
34	snake->part[i] = snake->head:
35	}
36	dir = RIGHT;
37	charlock = LEFT;
38	break;
39	}
40	}

15. Hàm void removeDesGate(): xóa cổng đích

Dòng	Code
1	void removeDesGate(int stage) {
2	switch(stage) {
3	case 1: case 2:
4	if (snake->part[snake->size - 1].x != LEFT_SIDE_X + 2 && snake->part[snake->size - 1].y != TOP_SIDE_Y + 2) {
5	cleargate = true;
6	drawBlank(LEFT_SIDE_X + 1, TOP_SIZE_Y + 1, 3, 3);
7	}
8	break;
9	case 3:

10	if (snake->part[snake->size - 1].x != LEFT_SIDE_X + 1 && snake->part[snake->size - 1].y != TOP_SIDE_Y + 4) {
11	cleargate = true;
12	drawBlank(LEFT_SIDE_X + 1, TOP_SIZE_Y + 3, 3, 3);
13	}
14	break;
15	case 4:
16	if (snake->part[snake->size - 1].x != RIGHT_SIDE_X - 5 && snake->part[snake->size - 1].y != TOP_SIDE_Y + 2) {
17	cleargate = true;
18	drawBlank(RIGHT_SIDE_X - 6, TOP_SIZE_Y + 1, 3, 3);
19	}
20	break;
21	case 5:
22	if (snake->part[snake->size - 1].x != LEFT_SIDE_X + 2 && snake->part[snake->size - 1].y != TOP_SIDE_Y + 14) {
23	cleargate = true;
24	drawBlank(LEFT_SIDE_X + 1, TOP_SIZE_Y + 13, 3, 3);
25	}
26	break;
27	}
28	}
29	}

16. Hàm void renderNewStage(): khởi tạo màn mới

Dòng	Code
1	void renderNewStage(int &gatecount, int &maxpoint, bool &cleargate) {
2	POSITION temp;
3	temp = snake->head;
4	drawBlank(27, 6, 89, 19);
5	stage++;
6	graphSet(stage % 5 + 1, snake);
7	gatecount = 0;
8	maxpoint += 100;
9	desGate(stage % 5 + 1);
10	cleargate = false;
11	gotoXY(temp.x, temp.y);
12	textColorWithBackground(map[temp.y - TOP_SIDE_Y][temp.x - LEFT_SIDE_X][1], WHITE);
13	if (map[temp.y - TOP_SIDE_Y][temp.x - LEFT_SIDE_X][0] == 0) cout << " ";
14	else cout << char(map[temp.y - TOP_SIDE_Y][temp.x - LEFT_SIDE_X][0]);
15	}

17. Hàm void resetScore(): cập nhật điểm

Dòng	Code
1	void restScore() {
2	textColorWithBackground(DARK_YELLOW, WHITE);
3	gotoXY(10, 9);
4	cout << score;
5	}

18. Hàm void resetSpeed(): cập nhật tốc độ rắn

Dòng	Code
1	void resetSpeed() {
2	textColorWithBackground(DARK_YELLOW, WHITE);
3	gotoXY(10, 9);
4	if (speedcontrol < 14)
5	cout << speedcontrol + 1;
6	else
7	cout << “MAX”;
8	}

VI. Xử lý va chạm và hiệu ứng khi va chạm

1. Hàm bool isValidSpawn(): kiểm tra vị trí khởi tạo

Dòng	Code
1	void isValidSpawn(int x, int y) {
2	if (x == snake->head.x && y == snake->head.y) return false;
3	for (int idx = 0; idx < snake->size; idx++) {
4	if (snake->part[idx].x == x && snake->part[idx].y == y) return false;
5	}
6	return true;
7	}

2. Hàm bool isValidFood(): kiểm tra vị trí in thức ăn

Dòng	Code
1	void isValidFood(int x, int y) {
2	if (x == snake->head.x && y == snake->head.y) return false;
3	for (int idx = 0; idx < snake->size; idx++) {
4	if (snake->part[idx].x == x && snake->part[idx].y == y) return false;
5	}
6	if (map[y - TOP_SIDE_Y][x - LEFT_SIDE_X][0] == BLOCK

7	|| map[y - TOP_SIDE_Y][x - LEFT_SIDE_X][0] == UPPER_BLOCK
8	|| map[y - TOP_SIDE_Y][x - LEFT_SIDE_X][0] == BOTTOM_BLOCK) return false;
9	return true;
10	}

3. Hàm bool isFoodInBush(): kiểm tra thức ăn có khởi tạo trùng vị trí của bụi hay không

Dòng	Code
1	bool isFoodInBush(POSITION food) {
2	if (map[food.y - TOP_SIDE_Y][food.x - LEFT_SIDE_X][0] <= BUSH_LV3 && map[food.y - TOP_SIDE_Y][food.x - LEFT_SIDE_X][0] >= BUSH_LV1) {
3	return true;
4	}
5	return false;
6	}

4. Hàm void checkCollisionBoard(): kiểm tra va chạm với các cạnh màn chơi

Dòng	Code
1	void checkCollisionBoard(SNAKE *snake) {
2	if (snake->head.x <= LEFT_SIDE_X || snake->head.x >= RIGHT_SIDE_X)
3	gameover = true;
4	if (snake->head.y <= TOP_SIDE_Y || snake->head.y >= BOTTOM_SIDE_Y)
5	gameover = true;
6	}

5. Hàm void checkCollisionObstacles(): kiểm tra va chạm với các vật thể trong màn

Dòng	Code
1	void checkCollisionObstacles(SNAKE *snake) {
2	if (map[snake->head.y - TOP_SIDE_Y][snake->head.x - LEFT_SIDE_X][0] == BLOCK
3	|| map[snake->head.y - TOP_SIDE_Y][snake->head.x - LEFT_SIDE_X][0] == UPPER_BLOCK

4	|| map[snake->head.y - TOP_SIDE_Y][snake->head.x - LEFT_SIDE_X][0] == BOTTOM_BLOCK)
5	gameover = true;
6	}

6. Hàm void checkSelfHitting(): kiểm tra va chạm với thân rắn

Dòng	Code
1	void checkCollisionObstacles(SNAKE *snake) {
2	for (int i = 0; i < snake->size; i++) {
3	if (snake->head.x == snake->part[i].x && snake->head.y == snake->part[i].y) gameover = true;
5	}
6	}

7. Hàm bool getInGate(): kiểm tra qua cổng

Dòng	Code
1	bool getInGate(POSITION pos) {
2	if (map[pos.y - TOP_SIDE_Y][pos.x - LEFT_SIDE_X][0] == GATE_SPOT) {
3	return true;
4	}
5	return false;
6	}

VII. Xử lý lưu (save) và tải (load) file dữ liệu (data) game

1. Hàm void saveClassicFile(): lưu dữ liệu chế độ chơi Classic

Dòng	Code
1	void saveClassicFile(CLASSICDATA* data, int playercount) {
2	ofstream fOut;
3	fOut.open("classicData.txt", ios::out);
4	fOut << playercount << endl;
5	
6	for (int i = 0; i < playercount; i++) {
7	fOut << "Player: " << data[i].classicname << endl;
8	fOut << "Stage: " << data[i].stage << endl;
9	fOut << "Score: " << data[i].score << endl;
10	fOut << "Difficulty: " << data[i].difficulty << endl;
11	fOut << "Head: " << data[i].snake->head.x << ' ' << data[i].snake-

	>head.y << endl;
12	fOut << "Part:";
13	int j = 0;
14	for (; j < data[i].snake->size - 1; j++) {
15	fOut << ' ' << data[i].snake->part[j].x << ' ' << data[i].snake->part[j].y << ','
16	}
17	fOut << ' ' << data[i].snake->part[j].x << ' ' << data[i].snake->part[j].y << '.' << endl;
18	fOut << endl;
19	}
20	fOut.close();
21	}

2. Hàm void loadClassicFile(): tải dữ liệu chế độ chơi Classic

Dòng	Code
1	void loadClassicFile(CLASSICDATA* data, int playercount) {
2	ifstream fIn;
3	fIn.open("classicData.txt", ios::in);
4	fIn >> playercount;
5	
6	if (playercount != 0) {
7	data = new CLASSICDATA[playercount];
8	for (int i = 0; i < playercount; i++) {
9	fIn.ignore(100, ' ');
10	getline(fIn, data[i].classicname);
11	fIn.ignore(100, ' ');
12	fIn >> data[i].stage;
13	fIn.ignore(100, ' ');
14	fin >> data[i].score;
15	fin.ignore(100, ':');
16	fin >> data[i].difficulty;
17	fin.ignore(100, ':');
18	data[i].snake = new SNAKE;
19	fIn >> data[i].snake->head.x >> data[i].snake->head.y;
20	fIn.ignore(100, ':');
21	data[i].snake->size = data[i].score / 10 + 1;
22	data[i].snake->part = new POSITION[data[i].snake->size];
23	for (int j = 0; j < data[i].snake->size; j++) {
24	fIn >> data[i].snake->part[j].x >> data[i].snake->part[j].y;
25	fIn.ingnore();
26	}
27	fIn.ignore(2);
28	}
29	}
30	fIn.close()
31	}

3. Hàm CLASSICDATA* pushClassicData(): lưu thông tin người chơi mới trong chế độ Classic

Dòng	Code
1	CLASSICDATA* pushClassicData(CLASSICDATA* data, CLASSICDATA player, int &playercount) {
2	CLASSICDATA* temp = new CLASSICDATA[playercount + 1];
3	if (data != nullptr) {
4	temp[0] = player;
5	for (int i = 0; i < playercount; i++) {
6	temp[i + 1] = data[i];
7	}
8	playercount++;
9	} else {
10	playercount++;
11	temp[0] = player;
12	}
13	return temp;
14	}

4. Hàm void freeClassicData(): Giải phóng bộ nhớ tạm thời trong chế độ Classic

Dòng	Code
1	void freeClassicData(CLASSICDATA* data, int playercount) {
2	for (int i = 0; i < playercount; i++) {
3	delete[] data[i].snake->part;
4	delete data[i].snake;
5	}
6	delete[] data;
7	}

5. Hàm void saveInfiniteFile(): lưu dữ liệu chế độ chơi Infinite

Dòng	Code
1	void saveInfiniteFile(INFINITEDATA* data, int playercount) {
2	ofstream fOut;
3	fOut.open("infiniteData.txt", ios::out);
4	fOut << playercount << endl;
5	
6	for (int i = 0; i < playercount; i++) {
7	fOut << "Player: " << data[i].infinitename<< endl;
8	fOut << "Score: " << data[i].score << endl;
9	fOut << "Head: " << data[i].snake->head.x << ' ' << data[i].snake->head.y << endl;

10	fOut << "Part:";
11	int j = 0;
12	for (; j < data[i].snake->size - 1; j++) {
13	fOut << ' ' << data[i].snake->part[j].x << ' ' << data[i].snake->part[j].y << ','
14	}
15	fOut << ' ' << data[i].snake->part[j].x << ' ' << data[i].snake->part[j].y << '.' << endl;
16	fOut << endl;
17	}
18	fOut.close();
19	}

6. Hàm void loadInfiniteFile(): tải dữ liệu chế độ chơi Infinite

Dòng	Code
1	void loadInfiniteFile(INFINITEDATA* data, int playercount) {
2	ifstream fIn;
3	fIn.open("infiniteData.txt", ios::in);
4	fIn >> playercount;
5	
6	if (playercoubt != 0) {
7	data = new INFINITEDATA[playercount];
8	for (int i = 0; i < playercount; i++) {
9	fIn.ignore(100, ' ');
10	getline(fIn, data[i].infinitename);
11	fIn.ignore(100, ' ');
12	fin >> data[i].score;
13	fin.ignore(100, ':');
14	data[i].snake = new SNAKE;
15	fIn >> data[i].snake->head.x >> data[i].snake->head.y;
16	fIn.ignore(100, ':');
17	data[i].snake->size = data[i].score / 10 + 1;
18	data[i].snake->part = new POSITION[data[i].snake->size];
19	for (int j = 0; j < data[i].snake->size; j++) {
20	fIn >> data[i].snake->part[j].x >> data[i].snake->part[j].y;
21	fIn.ingnore();
22	}
23	fIn.ignore(2);
24	}
25	}
26	fIn.close()
27	}

7. Hàm INFINITEDATA* pushInfiniteData(): lưu thông tin người chơi mới trong chế độ Infinite

Dòng	Code
1	INFINITEDATA* pushInfiniteData(INFINITEDATA* data, INFINITEDATA player, int &playercount) {
2	INFINITEDATA* temp = new INFINITEDATA[playercount + 1];
3	if (data != nullptr) {
4	temp[0] = player;
5	for (int i = 0; i < playercount; i++) {
6	temp[i + 1] = data[i];
7	}
8	playercount++;
9	} else {
10	playercount++;
11	temp[0] = player;
12	}
13	return temp;
14	}

8. Hàm void freeInfiniteData(): giải phóng bộ nhớ chế độ chơi Infinite

Dòng	Code
1	void freeInfiniteData(INFINITEDATA* data, int playercount) {
2	for (int i = 0; i < playercount; i++) {
3	delete[] data[i].snake->part;
4	delete data[i].snake;
5	}
6	delete[] data;
7	}

9. Hàm void saveTimeRushFile(): lưu dữ liệu chế độ chơi Time Rush

Dòng	Code
1	void saveTimeRushFile(TIMERUSHDATA* data, int playercount) {
2	ofstream fOut;
3	fOut.open(“timerushData.txt", ios::out);
4	fOut << playercount << endl;
5	
6	for (int i = 0; i < playercount; i++) {
7	fOut << "Player: " << data[i].timerushname<< endl;
8	fOut << "Stage: " << data[i].stage << endl;
9	fOut << "Score: " << data[i].score << endl;
10	fOut << "Difficulty: " << data[i].difficulty << endl;
11	fOut << "Head: " << data[i].snake->head.x << ' ' << data[i].snake->head.y << endl;
12	fOut << "Part:";
13	int j = 0;
14	for (; j < data[i].snake->size - 1; j++) {

15	fOut << ' ' << data[i].snake->part[j].x << ' ' << data[i].snake->part[j].y << ','
16	}
17	fOut << ' ' << data[i].snake->part[j].x << ' ' << data[i].snake->part[j].y << '.' << endl;
18	fOut << endl;
19	}
20	fOut.close();
21	}

10. Hàm void loadTimeRushFile(): tải dữ liệu chế độ chơi Time Rush

Dòng	Code
1	void loadTimeRushFile(TIMERUSHDATA* data, int playercount) {
2	ifstream fIn;
3	fIn.open("timeRushData.txt", ios::in);
4	fIn >> playercount;
5	
6	if (playercount != 0) {
7	data = new TIMERUSHDATA[playercount];
8	for (int i = 0; i < playercount; i++) {
9	fIn.ignore(100, ' ');
10	getline(fIn, data[i].timerushdata);
11	fIn.ignore(100, ' ');
12	fIn >> data[i].stage;
13	fIn.ignore(100, ' ');
14	fin >> data[i].score;
15	fin.ignore(100, ':');
16	fin >> data[i].difficulty;
17	fin.ignore(100, ':');
18	data[i].snake = new SNAKE;
19	fIn >> data[i].snake->head.x >> data[i].snake->head.y;
20	fIn.ignore(100, ':');
21	data[i].snake->size = data[i].score / 10 + 1;
22	data[i].snake->part = new POSITION[data[i].snake->size];
23	for (int j = 0; j < data[i].snake->size; j++) {
24	fIn >> data[i].snake->part[j].x >> data[i].snake->part[j].y;
25	fIn.ingnore();
26	}
27	fIn.ignore(2);
28	}
29	}
30	fIn.close()
31	}

11. Hàm TIMERUSHDATA* pushTimeRushData(): lưu thông tin người chơi mới trong chế độ Time Rush

Dòng	Code
1	TIMERUSHDATA* pushTimeRushData(TIMERUSHDATA* data, TIMERUSHDATAplayer, int &playercount) {
2	TIMERUSHDATA* temp = new TIMERUSHDATA[playercount + 1];
3	if (data != nullptr) {
4	temp[0] = player;
5	for (int i = 0; i < playercount; i++) {
6	temp[i + 1] = data[i];
7	}
8	playercount++;
9	} else {
10	playercount++;
11	temp[0] = player;
12	}
13	return temp;
14	}

12. Hàm void freeTimeRushData(): giải phóng bộ nhớ trong cho chế độ Time Rush

Dòng	Code
1	void freeTimeRushData(TIMERUSHDATA* data, int playercount) {
2	for (int i = 0; i < playercount; i++) {
3	delete[] data[i].snake->part;
4	delete data[i].snake;
5	}
6	delete[] data;
7	}

HẾT