

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN CUỐI KỲ
KỸ THUẬT LẬP TRÌNH

Giáo viên hướng dẫn: Trương Toàn Thịnh

Nhóm thực hiện: Nhóm 15

Lớp: 20CTT1

Niên khóa: 2020-2024

Thành phố Hồ Chí Minh, ngày 12 tháng 6 năm 2021

BÁO CÁO TỔNG KẾT ĐỒ ÁN CUỐI KỲ

KỸ THUẬT LẬP TRÌNH

ĐỀ TÀI: PHÁT TRIỂN GAME BĂNG QUA ĐƯỜNG



fit@hcmus

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM
KHOA CÔNG NGHỆ THÔNG TIN

MỤC LỤC

MỤC LỤC	3
LỜI GIỚI THIỆU	4
VỀ NHÓM THỰC HIỆN ĐỒ ÁN:	4
VỀ ĐỒ ÁN GAME BẰNG QUA ĐƯỜNG:	4
TÓM TẮT KỊCH BẢN GAME VÀ PHƯƠNG ÁN XÂY DỰNG GAME	5
CÁC CÔNG CỤ HỖ TRỢ PHÁT TRIỂN ĐỒ ÁN	6
CÁC YÊU CẦU ĐỒ ÁN VÀ TÓM TẮT PHƯƠNG ÁN GIẢI QUYẾT	7
CÁC BƯỚC XÂY DỰNG TRÒ CHƠI	9
PHẦN 1: XÂY DỰNG THƯ VIỆN CHỨA KHAI BÁO CÁC THƯ VIỆN CHUẨN CỦA C++, CÁC HẲNG SỐ, KIỂU DỮ LIỆU, BIẾN TOÀN CỤC VÀ HÀM CƠ BẢN (VARSANDLIBS.H).	9
PHẦN 2: XÂY DỰNG THƯ VIỆN CHỨA CÁC HÀM SỬ DỤNG ĐỂ QUẢN LÝ VÀ ĐIỀU KHIỂN CÁC CHỨC NĂNG KHI CHƠI GAME, CÁC HÀM TRANG TRÍ, HIỆU ỨNG CHO GAME (GAMECONTROL.H).	15
PHẦN 3: XÂY DỰNG THƯ VIỆN CHỨA HÀM XỬ LÝ CÁC THAO TÁC ĐƯỢC YÊU CẦU TỪ NGƯỜI CHƠI (GAMECOMMAND.H).	29
PHẦN 4: XÂY DỰNG THƯ VIỆN CHỨA CÁC HÀM HỖ TRỢ DI CHUYỂN NHÂN VẬT TRONG GAME (GAMEMOVEMENT.H).	32
PHẦN 5: XÂY DỰNG THƯ VIỆN CHỨA HÀM CHẠY CHO TIỂU TRÌNH, QUẢN LÝ NHỮNG SỰ KIỆN DIỄN RA TRÊN MÀN HÌNH CONSOLE (CONSOLECONTROL.H).	34
PHẦN 6: XÂY DỰNG HÀM MAIN VÀ FILE MÃ NGUỒN, SỬ DỤNG CÁC HÀM VÀ THƯ VIỆN ĐÃ XÂY DỰNG TRƯỚC ĐÓ (CROSSYROADPROJECT.CPP)	37
NGUỒN THAM KHẢO	43

LỜI GIỚI THIỆU

Về nhóm thực hiện đồ án:

Thông tin giáo viên hướng dẫn:

- Tiến sĩ Trương Toàn Thịnh – giáo viên bộ môn Công nghệ Phần mềm – trường Đại học Khoa học Tự nhiên
- Địa chỉ email liên lạc: ttthinh@fit.hcmus.edu.vn

Danh sách các thành viên của nhóm:

Họ và Tên	MSSV	Địa chỉ email liên lạc
Nguyễn Minh An	20120029	20120029@student.hcmus.edu.vn
Võ Hoài An	20120033	20120033@student.hcmus.edu.vn
Vương Lê Đức Bình	20120043	20120043@student.hcmus.edu.vn
Trần Hoàng Phương Nam	20120141	20120141@student.hcmus.edu.vn

Về đồ án game Băng Qua Đường:

Đây là đồ án cuối kì môn Kỹ thuật Lập trình nằm trong chương trình giảng dạy của khoa Công nghệ Thông tin – trường Đại học Khoa học Tự nhiên. Đồ án yêu cầu phối hợp các kỹ thuật và cấu trúc dữ liệu cơ bản để xây dựng một trò chơi đơn giản – Băng qua đường.

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

Phản hướng dẫn thực hiện đồ án được cung cấp bởi giáo viên hướng dẫn giúp sinh viên xây dựng trò chơi ở mức độ cơ bản. Sinh viên cần tự nghiên cứu để hoàn thiện đồ án một cách tốt nhất có thể.

Phần báo cáo sau đây là hướng dẫn cụ thể các bước xây dựng hoàn thiện đồ án của nhóm 15 – Lớp 20CTT1 – Khoa Công nghệ Thông tin – trường Đại học Khoa học Tự nhiên. Trong quá trình làm việc, khó tránh khỏi sai sót, cũng như những dòng lệnh chưa thật sự tường minh, rõ ràng (code smell), vì thế mọi góp ý thông qua địa chỉ email của các thành viên nhóm rất quan trọng và được trân trọng, qua đó nhóm có thể chỉnh sửa và hoàn thiện đồ án hơn nữa. Xin chân thành cảm ơn!

TÓM TẮT KỊCH BẢN GAME VÀ PHƯƠNG ÁN XÂY DỰNG GAME

- Băng qua đường là một trò chơi giải trí nổi tiếng với lối chơi vô cùng đơn giản. Trong trò chơi này, bạn sẽ điều khiển một nhân vật có hình dáng chữ ‘Y’ với nhiệm vụ là cố gắng băng qua một con đường lớn với dòng xe đông đúc và tấp nập (nhưng không kẹt xe giống Xa lộ Hà Nội mà chúng ta thường thấy) mà không gặp tai nạn.
- Game được thiết kế với 3 level cơ bản. Đồng nghĩa với việc bạn phải giúp lần lượt 3 người qua đường mà ở phía bên kia đường, họ không chạm vào nhau. Sau khi hoàn thành thử thách, trò chơi được đặt lại (trừ điểm số của bạn) và bạn tiếp tục giúp đỡ 3 người khác. Những chiếc xe trên con đường di chuyển với tốc độ tăng dần qua 3 level, hướng đi và chiều dài mỗi xe là ngẫu nhiên, đồng thời sau một khoảng thời gian nhất định (tùy vào level hiện tại) những chiếc xe sẽ tạm đứng yên trong khoảng 1s để người chơi có thể “thở” sau những pha tạt đầu xe kịch tính.
- Trong trường hợp có tai nạn đáng tiếc xảy ra, đó là va chạm giữa người và những chiếc xe điên cuồng hoặc giữa người đang qua đường và những người đã qua đường trước đó, rõ ràng bạn sẽ trở thành người thua cuộc và phải chơi lại từ đầu.
- Tại mỗi màn chơi, bạn được cộng 150 điểm thưởng, thế nhưng sau mỗi bước di chuyển, điểm của bạn sẽ bị trừ đi 1. Số điểm này được cộng dồn sau mỗi màn chơi và chỉ bị đặt lại khi nhân vật gặp tai nạn – người chơi thua cuộc.
- Trong quá trình chơi game, người chơi sử dụng các phím ‘W’, ‘A’, ‘S’, ‘D’ để di chuyển lên, qua trái, xuống, qua phải tương ứng. Sử dụng phím ‘P’ để tạm dừng game, phím ‘R’ hoặc bất kì phím di chuyển để tiếp tục chơi, phím ‘K’ để lưu lại game vào file cá nhân và phím ‘Esc’ để quay về màn hình chính (Start Menu) trong trường hợp bạn quá ức chế với trò chơi.
- Mở đầu game, màn hình chính sẽ có 4 khung gồm 4 sự lựa chọn:
 - “New Game”: Sử dụng phím ‘N’ để bắt đầu một trò chơi mới.
 - “Load Game”: Sử dụng phím ‘L’ để tải và tiếp tục phần chơi cũ nếu bạn đã lưu lại dữ liệu trước đó.
 - “Exit”: Sử dụng phím ‘Esc’ để thoát khỏi trò chơi.
 - “About”: Sử dụng phím ‘O’ để xem thông tin của nhóm sáng tạo ra game cùng hướng dẫn chơi game.

CÁC CÔNG CỤ HỖ TRỢ PHÁT TRIỂN ĐỒ ÁN

- Phần mềm lập trình hệ thống Visual Studio 2019
 - Visual Studio được tạo ra bởi Microsoft, sở hữu giao diện thân thiện, dễ dàng sử dụng cho người mới bắt đầu, được đưa vào chương trình giảng dạy như một công cụ để lập trình C/C++ tại HCMUS. Ngoài ra, Visual Studio còn có tính năng LiveShare, giúp các thành viên trong nhóm có thể theo dõi và quản lý code trực tiếp.
- Hệ thống quản lý dự án và phiên bản code Git và GitHub
 - Git và GitHub hỗ trợ trong vấn đề quản lý phiên bản là lưu trữ source code lên internet, dễ dàng chia sẻ và chỉnh sửa, góp ý giữa các thành viên trong nhóm. Ngoài ra, Git và GitHub cũng được tích hợp với Visual Studio khiến việc sử dụng rất dễ dàng.
- Công cụ vẽ sơ đồ draw.io (draw.io) (diagrams.net)
 - Công cụ hỗ trợ vẽ flowchart (lưu đồ) nhằm minh họa và tóm tắt các hàm và các vấn đề chương trình cần thực thi. Có thể tích hợp với GitHub.
- Phần mềm soạn thảo văn bản Microsoft Word
 - Hỗ trợ soạn thảo báo cáo đồ án. Có thể xuất sang định dạng PDF để không làm lỗi font và hình ảnh.
- Phần mềm trình chiếu Microsoft PowerPoint
 - Hỗ trợ tạo bài thuyết trình đồ án.
- Phần mềm đọc và chỉnh sửa file PDF Adobe Reader
 - Chỉnh sửa file báo cáo dạng PDF được tạo ra từ Word.
- Nền tảng họp trực tuyến Zoom
 - Hỗ trợ họp trực tuyến giữa các thành viên trong nhóm.

CÁC YÊU CẦU ĐỒ ÁN VÀ TÓM TẮT PHƯƠNG ÁN GIẢI QUYẾT

Trong quá trình xây dựng đồ án, ngoài những hướng dẫn đã có từ giảng viên, đồ án cũng có một số yêu cầu thêm, cần nhóm xây dựng xử lý được liệt kê dưới đây cùng với đó là tóm tắt phương án giải quyết của nhóm.

Xử lý va chạm với người băng qua đường trước đó

Trong hướng dẫn chưa xử lý việc ‘Y’ va chạm với các ‘Y’ đã về trước đó. Khi điều này xảy ra, ta cũng xử lý tạm dừng trò chơi và hỏi ý kiến người dùng xem có muốn tiếp tục hay không?

Phương án giải quyết: Tạo một mảng tĩnh *prev_pos[]* để lưu lại vị trí hoành độ *x* của nhân vật đã về đích trước đó. Do sau khi qua level 3 thì trò chơi được đặt lại nên ta chỉ cần một mảng tĩnh đủ để chứa 2 phần tử (vị trí của nhân vật tại level 1 và level 2).

Xử lý lưu trò chơi/tải trò chơi đã lưu

Sinh viên bổ sung chức năng khi người dùng nhấn phím ‘L’ thì xuất hiện dòng lệnh yêu cầu người dùng nhập tên tập tin cần lưu lại. Tương tự khi nhấn phím ‘T’ thì xuất hiện dòng lệnh yêu cầu người dùng nhập tên tập tin cần tải lên. Hướng dẫn: Sinh viên tự tổ chức cấu trúc tập tin để lưu dữ liệu biến toàn cục trong chương trình

Phương án giải quyết: Dùng thư viện *fstream* được cung cấp bởi C++, lưu vào file .txt các thông số cần thiết của người chơi hiện tại gồm: level, điểm của người chơi, thông tin mô tả những dòng xe (hướng di chuyển, chiều dài, vị trí), vị trí của những nhân vật đã về đích trước đó (nếu có), vị trí của nhân vật hiện tại. Đối với quá trình tải lại trò chơi, ta cho dừng thread, thực hiện in ra lần lượt các thông số được lưu rồi cho thread chạy (cho trò chơi tiếp tục chạy).

Xử lý tạm dừng các toa xe

Trong hướng dẫn, các xe di chuyển liên tục, sinh viên hãy hiệu chỉnh lại sao cho mỗi toa xe đều có thể dừng lại trong một khoảng thời gian tùy ý để giúp trò chơi dễ chơi hơn khi lên cấp.

Phương án giải quyết: Ta thiết lập một biến boolean *delay* = false. Trong quá trình cập nhật những chiếc xe di chuyển, chỉ khi nào biến *delay* = false thì ta mới cập nhật vị trí mới cho xe, khi đó, nhân vật hoàn toàn có thể di chuyển nhưng xe thì không. Sau một khoảng thời gian nhất định, ta cập nhật lại biến *delay*.

Xử lý hiệu ứng khi va chạm

Khi người qua đường va chạm xe thì tạo hiệu ứng đơn giản minh họa việc va chạm.

Phương án giải quyết: Dùng thread và in hiệu ứng.

Xử lý màn hình chính

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

Phương án giải quyết: Tạo một hàm vẽ màn hình chính startMenu() in ra các sự lựa chọn cho người chơi, khi người chơi chọn thì mới bắt đầu trò chơi.

CÁC BƯỚC XÂY DỰNG TRÒ CHƠI

Phần 1: Xây dựng thư viện chứa khai báo các thư viện chuẩn của C++, các hằng số, kiểu dữ liệu, biến toàn cục và hàm cơ bản (varsANDlibs.h).

Bước 1: Khai báo các thư viện chuẩn C++ cần thiết

Dòng	
1	#pragma once
2	#include <iostream>
3	#include <fstream>
4	#include <Windows.h>
5	#include <time.h>
6	#include <thread>
7	#include <conio.h>
8	#include <string>
9	#include <algorithm>

Tại dòng 1, ta dùng lệnh #pragma once để tránh các thư viện bị gọi nhiều lần ở nhiều file. Từ dòng 2 đến dòng 9 là các thư viện cần thiết để dùng các hàm như SuspendThread(), min(), max(), Sleep() và các kiểu dữ liệu lưu trữ như string, các hàm đọc ghi file như ifstream, ofstream.

Bước 2: Khai báo các biến hằng số cho chương trình

Dòng	
1	const int MAX_CAR = 21;
2	const int MIN_CAR_LENGTH = 20;
3	const int MAX_CAR_LENGTH = 40;
4	const int MAX_SPEED = 3;
5	const int PLAYGROUND_SECTION_HEIGHT = 24, INFO_SECTION_HEIGHT = 4;
6	const int WIDTH_CONSOLE = 118, HEIGHT_CONSOLE = 30;
7	const POINT DEFAULT_CHARACTER_POS = { 58, 24 };
8	const string DEFAULT_PLAYER_NAME = "Anonymous";

Tại dòng 1, ta khai báo biến MAX_CAR là số xe tối đa xuất hiện trên màn hình console (21 xe, mỗi dòng một xe). Tiếp đến tại dòng 2, ta có biến MIN_CAR_LENGTH và dòng 3, MAX_CAR_LENGTH ghi chiều dài tối thiểu và tối đa của từng xe. Biến MAX_SPEED là tốc độ tối đa của xe, cũng như level cao nhất có thể đạt được. Tiếp tục khai báo hằng số WIDTH_CONSOLE và HEIGHT_CONSOLE tức là độ rộng và độ cao màn hình console, nằm bên

trong là hằng số vùng trò chơi `PLAYGROUND_SECTION_HEIGHT` và thông tin người chơi `INFO_SECTION_HEIGHT` chia màn hình console làm 2 phần. Dòng thứ 7 và thứ 8, ta gọi tọa độ vị trí của nhân vật mặc định khi vào chơi, tên người chơi được mặc định khi chơi lần đầu là “Anonymous”.

Bước 3: Khai báo kiểu dữ liệu lưu thông tin từng xe và mảng để chứa thông tin của toàn bộ các xe

Dòng	
1	<code>struct CAR {</code>
2	<code> int direction;</code>
3	<code> int length;</code>
4	<code>};</code>
5	<code>CAR* carInfo;</code>
6	<code>int** carArray;</code>

Ta khởi tạo cấu trúc lưu trữ `CAR` gồm hai thông tin là *direction* và *length* tương ứng với ý nghĩa là hướng xe đi hiện tại và độ dài của xe. Một biến con trỏ *carInfo* có kiểu cấu trúc `CAR` dạng mảng động 1 chiều lưu thông tin của từng xe, do đó mỗi phần tử là một kiểu biến `CAR`. Tiếp đến ta tạo một mảng động 2 chiều của *carArray*, với `carArray[i][j] = x` cho biết rằng chiếc xe ở dòng thứ *i*, phần xe thứ *j* nằm ở vị trí *x* theo chiều rộng của console.

Bước 4: Khai báo các biến cục bộ

Dòng	
1	<code>POINT player_pos;</code>
2	<code>int speedUP = 0;</code>
3	<code>int direction, speed, step;</code>
4	<code>bool state;</code>
5	<code>int prevPos[5];</code>
6	<code>string player_name;</code>
7	<code>POINT ghost_pos;</code>
8	<code>int ghost_height = 5, ghost_width = 7;</code>
9	<code>string ghost_shape[5];</code>

Dòng 1 *player_pos* thể hiện vị trí hiện tại của người chơi, đến dòng 2 *speedUP* là biến hỗ trợ để tăng tốc xe khi bắt đầu di chuyển, dòng 3 ta gọi biến xác định hướng di chuyển của người chơi (W, A, S, D) (*direction*), biến tốc độ tương ứng level hiện tại (*speed*) và biến đếm các bước đã di chuyển (*step*). Dòng tiếp theo (*state*) cho ta biết trạng thái sống (1) hay chết (0) của người chơi. Dòng 5 là một mảng tĩnh lưu lại vị trí nhân vật đã về đích ở level trước. Dòng 6 là một chuỗi lưu lại tên của người chơi.

Ba dòng 7, 8, 9 khai báo các biến dùng để lưu lại vị trí, kích thước và hình dạng của “con ma” – một hiệu ứng sử dụng khi người chơi thua cuộc.

Bước 5: Khai báo hàm cố định màn hình console

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

Trong bước này ta sẽ cố định màn hình với kích thước thích hợp, làm điều này giúp tránh trường hợp người dùng tự co giãn màn hình sẽ gây khó khăn trong quá trình xử lý bằng cách gọi hàm `FixConsoleWindow()`.

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

Bước 6: Xây dựng hàm để giải phóng dữ liệu vùng nhớ trong ổ cứng

Dòng	
1	<code>void deleteCarData() {</code>
2	<code> if (carArray == NULL carInfo == NULL)</code>
3	<code> return;</code>
4	<code> for (int i = 0; i < MAX_CAR; i++)</code>
5	<code> delete[] carArray[i];</code>
6	<code> delete[] carArray;</code>
7	<code> delete[] carInfo;</code>
8	<code>}</code>

Do có một số biến được khai báo dạng con trỏ, nên khi người chơi thoát chương trình, ta cần giải phóng dữ liệu vùng nhớ trong ổ cứng bằng lệnh `delete` tại dòng 5, 6, 7. Trong trường hợp ta kiểm tra thấy một trong hai mảng đang ở vùng nhớ NULL (theo logic chương trình thì `carInfo = NULL` dẫn đến `carArray`

= *NULL*) thì ta dừng hàm nếu không sẽ gặp lỗi *RuntimeError* khi chạy lệnh *delete* thứ 2.

Bước 7: Xây dựng hàm kiểm tra xem một tệp tin có tồn tại hay không

Dòng	
1	<code>bool isFileExist(string fileName) {</code>
2	<code> fstream fileInput(fileName + ".txt");</code>
3	<code> return fileInput.good();</code>
4	<code>}</code>

Do mỗi khi người chơi muốn tải lại một game đã lưu trước đó, có khả năng file đó không tồn tại, vì thế ta cần xây dựng hàm kiểm tra xem tệp tin có tồn tại hay không, hàm sẽ trả về *false* nếu đọc tệp tin không thành công theo dòng 3.

Bước 8: Xây dựng hàm di chuyển con trỏ tới vị trí mong muốn trên màn hình console

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

Trong trò chơi sẽ có rất nhiều vị trí mà ta muốn in tại đó, vì vậy ta cần có khả năng di chuyển tới tất cả các vị trí trong màn hình console. Hàm này sẽ chuyển con trỏ đến vị trí (x, y) trên màn hình console, góc trên cùng bên trái bắt đầu là vị trí (0,0).

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

Bước 9: Xây dựng hàm chỉnh màu cho các kí tự được in ra console

Dòng	
1	<code>void setTextColor(int code) {</code>
2	<code>HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE);</code>
3	<code>SetConsoleTextAttribute(color, code);</code>
4	<code>}</code>

Hàm `setTextColor()` cho phép chỉnh màu các kí tự in ra trong màn hình console nhờ lệnh `SetConsoleTextAttribute(color, code)`. Hàm này nhằm mục đích trang trí game.

Bước 10: Xây dựng hàm để đặt lại các biến cục bộ về trạng thái mặc định

Dòng	
1	<code>void resetData() {</code>
2	<code>deleteCarData();</code>
3	<code>direction = 'D';</code>
4	<code>speed = 1;</code>
5	<code>step = 0;</code>
6	<code>player_name = DEFAULT_PLAYER_NAME;</code>
7	<code>player_pos = DEFAULT_CHARACTER_POS;</code>
8	<code>carArray = new int* [MAX_CAR];</code>
9	<code>carInfo = new CAR[MAX_CAR];</code>
10	<code>srand((unsigned)time(NULL));</code>
11	<code>for (int i = 0; i < MAX_CAR; i++) {</code>
12	<code>carInfo[i].direction = rand() % 2;</code>
13	<code>carInfo[i].length = MIN_CAR_LENGTH + rand() % (MAX_CAR_LENGTH - MIN_CAR_LENGTH + 1);</code>
14	<code>carArray[i] = new int[carInfo[i].length];</code>
15	<code>int temp = (rand() % (WIDTH_CONSOLE - carInfo[i].length)) + 1;</code>
16	<code>for (int j = 0; j < carInfo[i].length; j++)</code>
17	<code>carArray[i][j] = temp + j;</code>
18	<code>}</code>
19	<code>memset(prevPos, 0, sizeof(prevPos));</code>
20	<code>}</code>

Ta xây dựng hàm `resetData()` dùng trong trường hợp người chơi tạo game mới, khi đó ta cần đặt lại mọi thông số trong game. Đầu tiên ở dòng 2, ta xóa vùng nhớ lưu trữ dữ liệu xe để tránh xung đột về kích thước. Sau đó đặt lại các biến *direction*, *speed*, *step*, *player_name*, *player_pos* về giá trị mặc định – giá

trị hằng số từ dòng 3 đến dòng 7. Ta cũng sử dụng hàm `rand()` để tạo ra hướng cũng như chiều dài xe một cách ngẫu nhiên (dòng 11, 13, 14). Biến *temp* tại dòng 16 được khai báo cũng là để sinh một giá trị ngẫu nhiên đặt làm vị trí của xe thứ *i*. Cuối cùng ta reset lại mảng *prevPos* bằng hàm `memset()` tại dòng 19.

Phần 2: Xây dựng thư viện chứa các hàm sử dụng để quản lý và điều khiển các chức năng khi chơi game, các hàm trang trí, hiệu ứng cho game (gameControl.h).

Bước 1: Xây dựng hàm kiểm tra va chạm

Dòng	
1	<code>bool isImpact(const POINT& p) {</code>
2	<code>if (player_pos.y == 2 player_pos.y == 24)</code>
3	<code>return false;</code>
4	<code>for (int i = 0; i < carInfo[player_pos.y - 3].length; i++)</code>
5	<code>if (p.x == carArray[player_pos.y - 3][i])</code>
6	<code>return true;</code>
7	<code>return false;</code>
8	<code>}</code>

Như đã mô tả ở trên, người chơi sẽ thua khi va chạm với xe hoặc nhân vật về đích trước đó, vì thế trước tiên ta cần xây dựng hàm kiểm tra xem nhân vật hiện tại có va chạm với xe nào không. Khi người chơi đang đứng ở vị trí $p.y$ tương ứng với xe thứ $y - 3$ (do yếu tố trang trí khi in ra màn hình). Ta kiểm tra vị trí hoành độ $p.x$ của nhân vật đấy có trùng với bất kỳ đoạn nào của xe tương ứng hay không (dòng 5). Nếu có trả về True (đã va chạm) và ngược lại. Tại dòng 2 có hai trường hợp được thêm vào, đó là khi nhân vật đang đứng ở vị trí bắt đầu và vị trí kết thúc thì không va chạm gì cả, trả về False.

Bước 2: Xây dựng hàm xử lý và in hiệu ứng khi người chơi thua cuộc

Dòng	
1	<code>void processDead() {</code>
2	<code>ghost_pos.y = max(player_pos.y, 8);</code>
3	<code>ghost_pos.x = min(player_pos.x, WIDTH_CONSOLE - 8);</code>
4	<code>setTextColor(8);</code>
5	<code>for (int i = PLAYGROUND_SECTION_HEIGHT; i > ghost_pos.y; --i) {</code>
6	<code>fillBox(1, i, 4, 0, "X_X");</code>
7	<code>fillBox(4, i, WIDTH_CONSOLE - 7, 0, " X_X");</code>
8	<code>}</code>
9	<code>while (ghost_pos.y > 5) {</code>
10	<code>fillBox(1, ghost_pos.y + 1, 4, 0, "X_X");</code>
11	<code>fillBox(4, ghost_pos.y + 1, WIDTH_CONSOLE - 7, 0, " X_X");</code>
12	<code>for (int i = ghost_height - 1; i >= 0; i--) {</code>

13	<code>fillBox(1, ghost_pos.y - i, WIDTH_CONSOLE - 2, 0, "</code>
14	<code>");</code>
15	<code>GotoXY(ghost_pos.x, ghost_pos.y - i);</code>
16	<code>setTextColor(6);</code>
17	<code>cout << ghost_shape[ghost_height - i - 1];</code>
18	<code>}</code>
19	<code>ghost_pos.y--;</code>
20	<code>Sleep(75);</code>
21	<code>}</code>
22	<code>while (ghost_pos.y > 0) {</code>
23	<code>fillBox(1, ghost_pos.y + 1, 4, 0, "X_X");</code>
24	<code>fillBox(4, ghost_pos.y + 1, WIDTH_CONSOLE - 7, 0, " X_X");</code>
25	<code>for (int i = ghost_pos.y - 2; i >= 0; i--) {</code>
26	<code>fillBox(1, ghost_pos.y - i, WIDTH_CONSOLE - 2, 0, "</code>
27	<code>");</code>
28	<code>GotoXY(ghost_pos.x, ghost_pos.y - i);</code>
29	<code>setTextColor(6);</code>
30	<code>cout << ghost_shape[ghost_height - i - 1];</code>
31	<code>}</code>
32	<code>ghost_pos.y--;</code>
33	<code>Sleep(100);</code>
34	<code>}</code>
35	<code>fillBox(1, 1, WIDTH_CONSOLE - 2, HEIGHT_CONSOLE - 3, " ");</code>
36	<code>Sleep(250);</code>
37	<code>endGameMenu();</code>
	<code>setTextColor(7);</code>
	<code>}</code>

Khi người chơi thua cuộc, biến *ghost_pos* ngay lập tức lưu lại vị trí lúc người chơi va chạm (sau đây được hiểu là va chạm với cả xe lẫn nhân vật về đích trước đó). Hiệu ứng thua là 1 hồn ma nạn nhân từ từ di chuyển lên trên vị trí đích đến và ra khỏi vùng chơi (playground). Dòng 2, 3 dùng để tinh chỉnh lại vị trí của hồn ma để khi in ra, nó không chen vào các khung đã được vẽ trước đó. Từ dòng 5 đến 8, hàm *fillBox()* sẽ được đề cập bên dưới sẽ lấp đầy màn hình vùng chơi bằng chuỗi X_X. Từ dòng 9 đến dòng 32 là quá trình di chuyển hồn ma từ vị trí nhân vật va chạm đến vị trí đích và bay mất. Hàm *Sleep()* được dùng để làm chậm quá trình này tạo tính thẩm mỹ. Sau đó, hàm *endGameMenu()* (sẽ được giải thích sau) được gọi ra, cuối hàm này là lệnh cập nhật biến *state = 0* cho chương trình biến rằng nhân vật đã chết và ngừng việc di chuyển xe.

Bước 3: Xây dựng hàm xử lý khi người chơi qua màn (lên level mới).

Dòng	
1	<code>void processPass(POINT& p) {</code>
2	<code> prevPos[speed - 1] = p.x;</code>
3	<code> if (speed == MAX_SPEED) {</code>
4	<code> for (int i = 0; i < 3; ++i) {</code>
5	<code> GotoXY(prevPos[i], 2);</code>
6	<code> cout << " ";</code>
7	<code> }</code>
8	<code> speed = 1;</code>
9	<code> }</code>
10	<code> else {</code>
11	<code> speed++;</code>
12	<code> for (int i = 0; i < MAX_CAR; ++i)</code>
13	<code> carInfo[i].direction = rand() % 2;</code>
14	<code> }</code>
15	<code> p = DEFAULT_CHARACTER_POS;</code>
16	<code> direction = 'D';</code>
17	<code>}</code>

Khi nhân vật được người chơi điều khiển chạm vạch đích, trước tiên tại dòng 2 ta luôn lưu lại vị trí hoành độ của nhân vật hiện tại vào mảng *prev_pos*. Sau đó có 2 trường hợp xảy ra. Một là người chơi ấy hiện đạt ở level < MAX_LEVEL, khi ấy ta chỉ cần tăng level lên 1 đơn vị (*speed++*, dòng 11) và cập nhật lại hướng di chuyển mới cho từng xe (tăng độ khó). Đối với trường hợp người chơi đã đạt MAX_LEVEL, ta xoá các vị trí nhân vật đã về đích cũ trên console và đặt lại level *speed* = 1 (dòng 8). Kết thúc việc xét hai trường hợp đã nêu, đến dòng 15 ta chuyển nhân vật mới về vị trí mặc định, sẵn sàng cho màn chơi mới.

Bước 4: Xây dựng hàm in ra nhân vật trên console

Dòng	
1	<code>void drawCharacter(const POINT& p, string s) {</code>
2	<code> setTextColor(6);</code>
3	<code> GotoXY(p.x, p.y);</code>
4	<code> cout << s;</code>
5	<code> setTextColor(7);</code>
6	<code>}</code>

Ta xây dựng hàm xuất kí tự thể hiện nhân vật ra console tại vị trí yêu cầu ($p.x, p.y$), đồng thời chỉnh nhân vật thành màu vàng (color code: 6).

Bước 5: Xây dựng hàm in ra những chiếc xe trên màn hình console

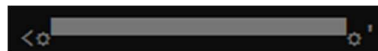
Dòng	
1	<code>void drawCars() {</code>
2	<code> setTextColor(8);</code>
3	<code> for (int i = 0; i < MAX_CAR; i++) {</code>
4	<code> string curCar(carInfo[i].length, char(223));</code>
5	<code> curCar[1] = char(15);</code>
6	<code> curCar[curCar.length() - 2] = char(15);</code>
7	<code> if (carInfo[i].direction == 1) {</code>
8	<code> curCar[0] = char(96);</code>
9	<code> curCar[curCar.length() - 1] = char(62);</code>
10	<code> }</code>
11	<code> else {</code>
12	<code> curCar[0] = char(60);</code>
13	<code> curCar[curCar.length() - 1] = char(39);</code>
14	<code> }</code>
15	<code> for (int j = 0; j < carInfo[i].length; j++) {</code>
16	<code> GotoXY(carArray[i][j], i + 3);</code>
17	<code> cout << curCar[j];</code>
18	<code> }</code>
19	<code> }</code>
20	<code> setTextColor(7);</code>
21	<code>}</code>

Ta xây dựng hàm `drawCars()` để in những chiếc xe ra màn hình với những thông tin kèm theo: hướng xe chạy, chiều dài xe, kí tự biểu thị xe.

Để biết kí tự biểu thị xe, ta cần xác định được hướng xe di chuyển. Nếu xe đi qua bên phải sẽ được biểu thị như sau:



Và ngược lại đối với bên trái:



Ta tạo ra chuỗi biểu thị xe bằng cách tạo mỗi string có độ dài bằng với độ dài xe bằng dòng 4 và thay đổi các kí tự sao cho phù hợp (dòng 5 đến dòng 14). Cuối cùng, dùng hàm `GotoXY()` đã xây dựng để di chuyển tới vị trí in ra từng đoạn xe (dòng 15 đến 18).

Bước 6: Xây dựng hàm cập nhật vị trí cho xe

Dòng	
1	<code>void moveCars() {</code>
2	<code> for (int i = 0; i < MAX_CAR; i++) {</code>
3	<code> if (carInfo[i].direction == 1) {</code>
4	<code> speedUP = 0;</code>
5	<code> do {</code>
6	<code> speedUP++;</code>
7	<code> for (int j = 0; j < carInfo[i].length - 1; j++)</code>
8	<code> {</code>
9	<code> carArray[i][j] = carArray[i][j + 1];</code>
10	<code> } while (carArray[i][carInfo[i].length - 1] + 1 ==</code>
11	<code> WIDTH_CONSOLE ? carArray[i][carInfo[i].length - 1] = 1 :</code>
12	<code> carArray[i][carInfo[i].length - 1]++;</code>
13	<code> } while (speedUP < speed);</code>
14	<code> }</code>
15	<code> if (carInfo[i].direction == 0) {</code>
16	<code> speedUP = 0;</code>
17	<code> do {</code>
18	<code> speedUP++;</code>
19	<code> for (int j = carInfo[i].length - 1; j > 0; j--)</code>
20	<code> {</code>
21	<code> carArray[i][j] = carArray[i][j - 1];</code>
22	<code> } while (carArray[i][0] - 1 == 0 ? carArray[i][0] =</code>
23	<code> WIDTH_CONSOLE - 1 : carArray[i][0]--;</code>
24	<code> } while (speedUP < speed);</code>
	<code> }</code>
	<code> }</code>

Ta cần xây dựng hàm cập nhật vị trí mới các toa xe là vì xe cần di chuyển, còn hàm `drawCars()` chỉ in xe tĩnh. Vì thế ta chỉ cần cập nhật vị trí mới, và để hàm `drawCars()` xuất ra vị trí mới liên tục, do tốc độ xử lý của chương trình, nhìn từ phía người chơi trông giống như xe di chuyển. Tùy vào hướng xe di chuyển ta cập nhật hai hướng khác nhau. Nhìn chung, với mỗi lần cập nhật, mỗi phần của xe (toa xe) sẽ nhận vị trí của toa đi trước, khi hàm `drawCars()` được gọi thì toa xe này đã di chuyển lên trước (dòng 8, 18). Biến `speedUP` tỏ ra hữu dụng khi cho ta khả năng update vị trí liên tục. VD: Khi `speed = 1`, thì `speedUP` sẽ tạo vòng lặp 1 lần, cập nhật vị trí toa xe lên một đơn vị, khi `speed = 2`, `speedUP` sẽ tạo vòng lặp 2 lần, cập nhật vị trí toa xe nhích lên 2 đơn vị, tạo cảm giác xe chạy nhanh hơn. Ngoài ra, cần xử lý trường hợp đầu xe chạm biên

trái-phải, khi đó vị trí đầu xe sẽ cập nhật thành vị trí đầu tiên ở phía đối diện (1 hoặc `WIDTH_CONSOLE - 1`) (dòng 10, 20).

Bước 7: Xây dựng hàm xoá các xe ở vị trí cũ sau khi cập nhật vị trí bằng hàm `moveCars()`

Dòng	
1	<code>void eraseCars() {</code>
2	<code> for (int i = 0; i < MAX_CAR; i++) {</code>
3	<code> if (carInfo[i].direction == 0) {</code>
4	<code> speedUP = 0;</code>
5	<code> do {</code>
6	<code> GotoXY(carArray[i][carInfo[i].length - 1 -</code> <code>speedUP], i + 3);</code>
7	<code> cout << " ";</code>
8	<code> speedUP++;</code>
9	<code> } while (speedUP < speed);</code>
10	<code> }</code>
11	<code> if (carInfo[i].direction == 1) {</code>
12	<code> speedUP = 0;</code>
13	<code> do {</code>
14	<code> GotoXY(carArray[i][0 + speedUP], i + 3);</code>
15	<code> cout << " ";</code>
16	<code> speedUP++;</code>
17	<code> } while (speedUP < speed);</code>
18	<code> }</code>
19	<code> }</code>
20	<code>}</code>

Sau khi cập nhật và in ra vị trí mới, ta gặp một vấn đề là phần đuôi xe ở vị trí cũ do không có kí tự nào ghi đè lên (vì đuôi xe ở vị trí mới đã nhích lên trước) nên nó vẫn sẽ hiện diện ở đây. Vì thế ta xây dựng hàm này để xoá phần đuôi thừa đây. Không cần xoá toàn bộ xe rồi vẽ lại, ta chỉ cần xoá đi phần đuôi xe thừa ra, ta hoàn toàn có thể biết được độ dài và dễ dàng xoá nhờ vào biến `speed` và `speedUP` (dòng 6, 7, 8 và dòng 14, 15, 16).

Bước 8: Xây dựng hàm lấp đầy một khoảng trống hình chữ nhật bằng một chuỗi kí tự (hàm `Trang trí`)

Dòng	
1	<code>void fillBox(int x, int y, int width, int height, string s) {</code>
2	<code> setTextColor(3);</code>
3	<code> for (int i = x; i <= x + width; i += int(s.length())) {</code>
4	<code> for (int j = y; j <= y + height; ++j) {</code>

5	GotoXY(i, j);
6	cout << s;
7	}
8	}
9	}



Hàm fillBox() dùng để lấp đầy khung hình chữ nhật có chiều dài *height* và chiều rộng *width* trên màn hình console, bắt đầu từ tọa độ (x, y) đến (x+width, y+height) (dòng 3, 4) với chuỗi s (dòng 6). Hàm này hiệu quả trong việc xoá trắng một phần màn hình console với s = “ ” hoặc trang trí cho hàm processDead() với s = “X_X”





Bước 9: Xây dựng hàm vẽ một bảng kích thước 1x2 (hàm Trang trí)

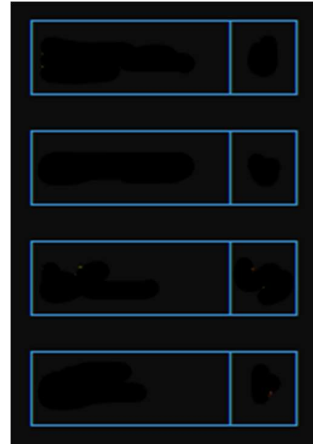
Dòng	
1	void drawSelectBox(int x, int y, int width, int height) {
2	setTextColor(3);
3	for (int i = x; i < x + width; ++i) {
4	GotoXY(i, y);
5	cout << char(196);
6	GotoXY(i, y + height);
7	cout << char(196);
8	}
9	for (int i = y; i < y + height; ++i) {
10	GotoXY(x, i);
11	cout << char(179);
12	GotoXY(x + width - 4, i);
13	cout << char(179);
14	GotoXY(x + width, i);
15	cout << char(179);
16	}
17	GotoXY(x, y); cout << char(218);
18	GotoXY(x + width - 4, y); cout << char(194);
19	GotoXY(x, y + height); cout << char(192);
20	GotoXY(x + width - 4, y + height); cout << char(193);
21	GotoXY(x + width, y); cout << char(191);
22	GotoXY(x + width, y + height); cout << char(217);
23	setTextColor(7);
24	}

Ý tưởng xây dựng hàm này nhằm tạo ra một bảng biểu gồm một bên nội dung và một bên là từ khoá tương ứng. Giúp người chơi có thể đối chiếu và

chọn từ khoá phù hợp. Hàm vẽ lên màn hình console một bảng 1x2 với chiều rộng là *width*, chiều dài là *height*. Phần ô từ khoá chiếm diện tích $4 * height$, còn lại là phần diện tích của ô từ khoá. Nguyên lý là đi đến vị trí cần vẽ và in ra kí tự phù hợp. Hàm trên sử dụng các kí tự sau:

- char(196) với kí tự  (dòng 5, 7) – đây là biên ngang
- char(179) với kí tự  (dòng 11, 13, 15) – đây là biên dọc
- Và các kí tự tương ứng để vẽ các góc vuông của bảng (dòng 17 đến dòng 22)

- Char(218) 
- Char(192) 
- Char(191) 
- Char(217) 
- Char(193) 
- Char(194) 



Ví dụ khi sử dụng hàm drawSelectBox() 4 lần

Bước 10: Xây dựng hàm in ra thông tin về nhóm sáng tạo game, thông tin về đồ án và hướng dẫn chơi game (hàm *Trang trí*)

Dòng	
1	<code>void drawAbout() {</code>
2	<code> setTextColor(6);</code>
3	<code> GotoXY(40, 7); cout << "DO AN KY THUAT LAP TRINH: GAME BANG QUA</code>
4	<code>DUONG";</code>
5	<code> setTextColor(14);</code>
6	<code> GotoXY(65, 9); cout << "Giao vien huong dan: TRUONG TOAN THINH";</code>
7	<code> setTextColor(6);</code>
8	<code> GotoXY(25, 11); cout << "Thuc hien boi: Nhom 15 - 20CTT1 - FIT -</code>
9	<code>HCMUS";</code>
10	<code> setTextColor(14);</code>
11	<code> GotoXY(40, 12); cout << "+ Nguyen Minh An\t\t\t20120029";</code>
12	<code> GotoXY(40, 13); cout << "+ Vo Hoai An\t\t\t20120033";</code>
13	<code> GotoXY(40, 14); cout << "+ Vuong Le Duc Binh \t\t\t20120043";</code>
14	<code> GotoXY(40, 15); cout << "+ Tran Hoang Phuong Nam\t\t\t20120141";</code>
15	<code> setTextColor(3);</code>

14	GotoXY(20, 17); cout << "Source Code: https://tinyurl.com/Team15KTLT ";
15	
16	GotoXY(20, 19); cout << "Just use W, A, S, D to move and you'll figure it out how to play. Have fun!!!";
17	setTextColor(7);
18	}

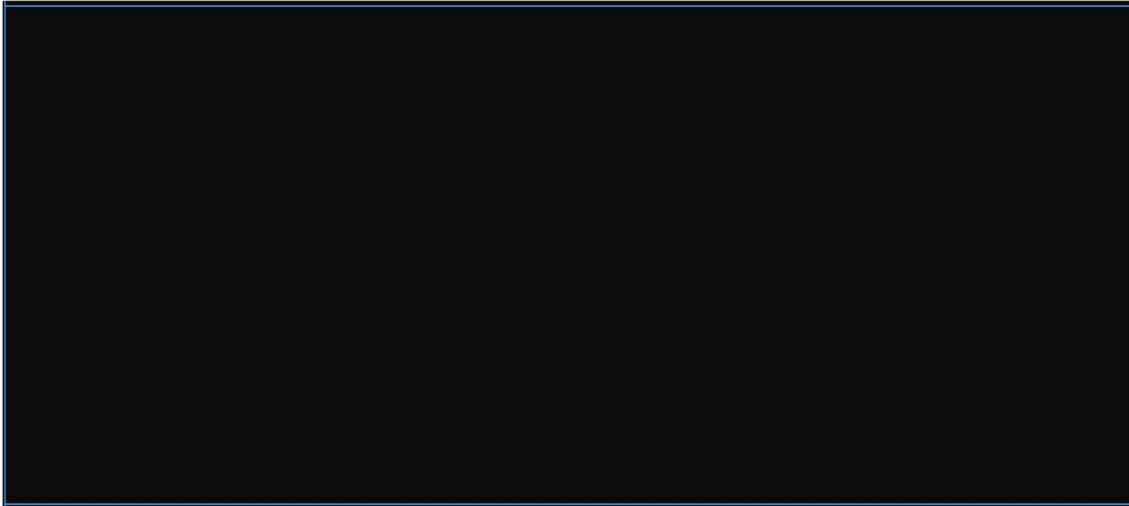
Bước 11: Xây dựng hàm vẽ khung viền hình chữ nhật (hàm Trang trí)

1	void drawBox(int x, int y, int width, int height) {
2	setTextColor(3);
3	GotoXY(x, y);
4	if (y == 0) {
5	cout << char(218);
6	for (int i = 1; i <= 52; ++i)
7	cout << char(196);
8	cout << " Crossy Road ";
9	for (int i = 1; i <= 52; ++i)
10	cout << char(196);
11	cout << char(191) << "\n";
12	y++;
13	}
14	GotoXY(x, y);
15	cout << char(195);
16	for (int i = x + 1; i < x + width; i++)
17	cout << char(196);
18	cout << char(180);
19	GotoXY(x, height + y);
20	cout << char(192);
21	for (int i = x + 1; i < x + width; i++)
22	cout << char(196);
23	cout << char(217);
24	for (int i = y + 1; i < height + y; i++) {
25	GotoXY(x, i);
26	cout << char(179);
27	GotoXY(x + width, i);
28	cout << char(179);
29	}
30	setTextColor(7);
31	}

Tại đây, ta xây dựng một hàm vẽ một phần giao diện của trò chơi, đó là phần khung viền hình chữ nhật. Phần khung viền này có tác dụng để định hình cho người chơi biết giới hạn của vùng di chuyển, cũng như để bao quanh trang trí cho phần thông số người chơi. Một điều đặc biệt ở hàm này là nếu vị trí bắt đầu vẽ (x, y) của hàm trùng với vị trí $(x, 0)$ thì nó luôn in ra tựa đề game như hình dưới đây (dòng 3).

Crossy Road

Phần tựa đề của game



Phần khung viền được vẽ ra (Từ dòng 7 đến dòng 23)

Bước 12: Xây dựng hàm in ra các chỉ số riêng của người chơi (hàm Trang trí)

Dòng	
1	<code>void drawInfo() {</code>
2	<code> setTextColor(6);</code>
3	<code> GotoXY(12, HEIGHT_CONSOLE - 3); cout << " ";</code>
4	<code> GotoXY(12, HEIGHT_CONSOLE - 2); cout << " ";</code>
5	<code> GotoXY(12, HEIGHT_CONSOLE - 3); cout << speed;</code>
6	<code> GotoXY(12, HEIGHT_CONSOLE - 2); cout << 150 * speed - step;</code>
7	<code> setTextColor(7);</code>
8	<code>}</code>

Tại dòng 3, 4, ta cần xoá trắng những kí tự hiện đang nằm ở vị trí cần in chỉ số, tránh trường hợp có kí tự thừa. Sau đó ta in ra level hiện tại của người chơi, đại diện bằng thông số *speed* (dòng 5) và điểm số người chơi, tính bằng *speed* và *step* (dòng 6).

Bước 13: Xây dựng hàm vẽ ra dòng nội dung giải thích về thông tin người chơi (hàm *Trang trí*)

Dòng	
1	<code>void drawGameMenu() {</code>
2	<code> setTextColor(3);</code>
3	<code> fillBox(1, HEIGHT_CONSOLE - 2, 50, 0, " ");</code>
4	<code> GotoXY(5, HEIGHT_CONSOLE - 4); cout << "Player Name: " << player_name;</code>
5	<code> GotoXY(5, HEIGHT_CONSOLE - 3); cout << "Level: " << speed;</code>
6	<code> GotoXY(5, HEIGHT_CONSOLE - 2); cout << "Score: " << 150 * speed - step;</code>
7	
8	<code> for (int i = 2; i < INFO_SECTION_HEIGHT + 1; ++i) {</code>
9	<code> GotoXY(40, HEIGHT_CONSOLE - i);</code>
10	<code> cout << char(179);</code>
11	<code> GotoXY(64, HEIGHT_CONSOLE - i);</code>
12	<code> cout << char(179);</code>
13	<code> GotoXY(86, HEIGHT_CONSOLE - i);</code>
14	<code> cout << char(179);</code>
15	<code> }</code>
16	<code> GotoXY(41, HEIGHT_CONSOLE - 3); cout << " Move " << char(16) << " W, A, S, D ";</code>
17	<code> GotoXY(65, HEIGHT_CONSOLE - 4); cout << " Pause Game " << char(16) << " P";</code>
18	<code> GotoXY(65, HEIGHT_CONSOLE - 3); cout << " Resume Game " << char(16) << " R";</code>
19	<code> GotoXY(65, HEIGHT_CONSOLE - 2); cout << " Save Game " << char(16) << " K";</code>
20	<code> GotoXY(87, HEIGHT_CONSOLE - 3); cout << " Return to Main Menu " << char(16) << " Esc ";</code>
21	<code> setTextColor(7);</code>
22	<code>}</code>

Khi xây dựng xong hàm `drawInfo()`, hàm đây chỉ in ra chỉ số mà không có lời giải thích ý nghĩa của chỉ số. Vì thế ta thiết kế một hàm in ra những lời giải thích đầy (dòng 4 đến dòng 6), kèm theo là các hướng dẫn sử dụng từ khoá khi chơi (dòng 16 đến dòng 20). Từ dòng 8 đến dòng 15 dùng để in ra các cột dọc chia cắt các nội dung cho dễ nhìn.

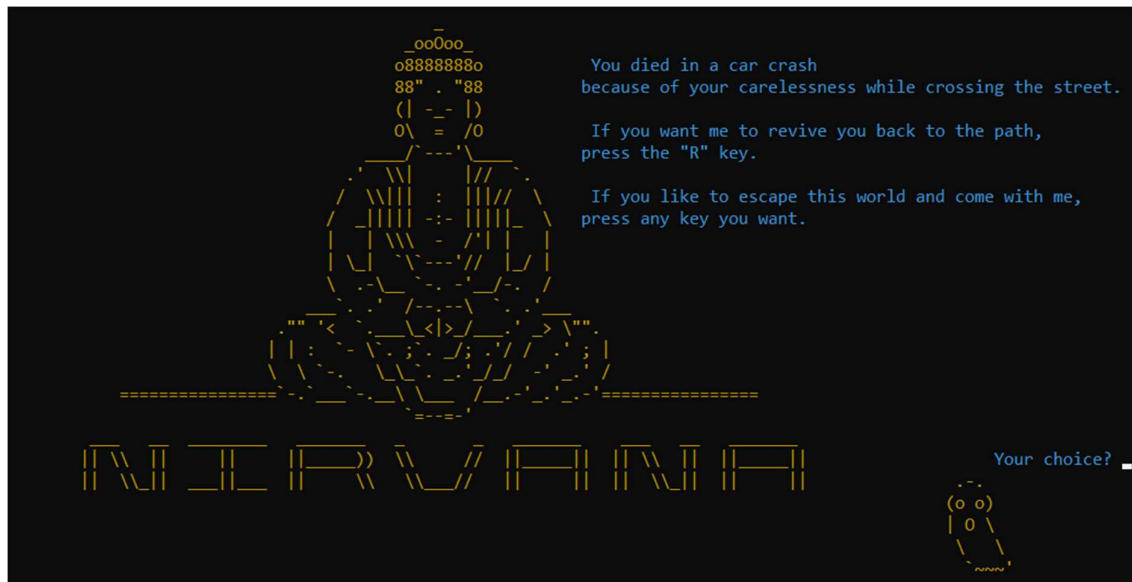
Bước 14: Xây dựng hàm vẽ hiệu ứng đơn giản khi người chơi thua cuộc và hỏi ý kiến người chơi về hành động tiếp theo (hàm *Trang trí*)

Dòng	
1	<code>void endGameMenu() {</code>
2	<code> setTextColor(6);</code>

3	GotoXY(45, 2); cout << "_";
4	GotoXY(42, 3); cout << "_ooOoo_";
5	GotoXY(41, 4); cout << "o8888888o";
6	GotoXY(41, 5); cout << "88\" . \"88\";
7	GotoXY(41, 6); cout << "(_-)";
8	GotoXY(41, 7); cout << "0\\ = /0\";
9	GotoXY(38, 8); cout << "____/'---'_____\";
10	GotoXY(36, 9); cout << ". ' \\\\ // `.\";
11	GotoXY(35, 10); cout << "/ \\\\ : // \\\";
12	GotoXY(34, 11); cout << "/ _ -:- _ \\\";
13	GotoXY(34, 12); cout << " \\\\\\\\ - /' \";
14	GotoXY(34, 13); cout << " _ \\`---'// _ \";
15	GotoXY(34, 14); cout << "\\ _.-_ `-.-'/_-./\";
16	GotoXY(32, 15); cout << "___`. .' /--.--\\ `.'. '__\";
17	GotoXY(29, 16); cout << "\"\" '< `.__\\< >/__.' _> \\\"\"\".\";
18	GotoXY(28, 17); cout << " : `'- \\`. ;`. _/; .' / / .' ; \";
19	GotoXY(28, 18); cout << "\\ \\ `-. _\\`_. _.'_/ / -' _.' /\";
20	GotoXY(13, 19); cout << "=====``._`-_\\ _\\ /_.-'. '.-.'=====\";
21	GotoXY(42, 20); cout << "`-----\";
22	GotoXY(9, 21); cout << " _____ - - _____\";
23	GotoXY(9, 22); cout << " \\\\ ____)) \\\\ // ____ \\\\ ____ \";
24	GotoXY(9, 23); cout << " _ __ __ \\\\ _// _ \";
25	
26	ghost_pos.y = HEIGHT_CONSOLE - 3;
27	for (int i = 0; i < 5; ++i) {
28	fillBox(1, ghost_pos.y + 1, WIDTH_CONSOLE - 2, 0, " ");
29	for (int j = i; j >= 0; j--) {
30	GotoXY(WIDTH_CONSOLE - i - ghost_width - 10, ghost_pos.y - j);
31	setTextColor(6);
32	cout << ghost_shape[i - j];
33	}
34	Sleep(150);
35	}
36	setTextColor(3);
37	GotoXY(60, 4); cout << " You died in a car crash \";
38	GotoXY(60, 5); cout << "because of your carelessness while crossing the street.\";
39	Sleep(500);
40	GotoXY(60, 7); cout << " If you want me to revive you back to the path,\";

41	GotoXY(60, 8); cout << "press the \"R\" key.";
42	GotoXY(60, 10); cout << " If you like to escape this world and come with me, ";
43	GotoXY(60, 11); cout << "press any key you want.";
44	Sleep(500);
45	GotoXY(WIDTH_CONSOLE - 16, ghost_pos.y - 5); cout << "Your choice?"
46	state = 0;
47	setTextColor(7);
48	}

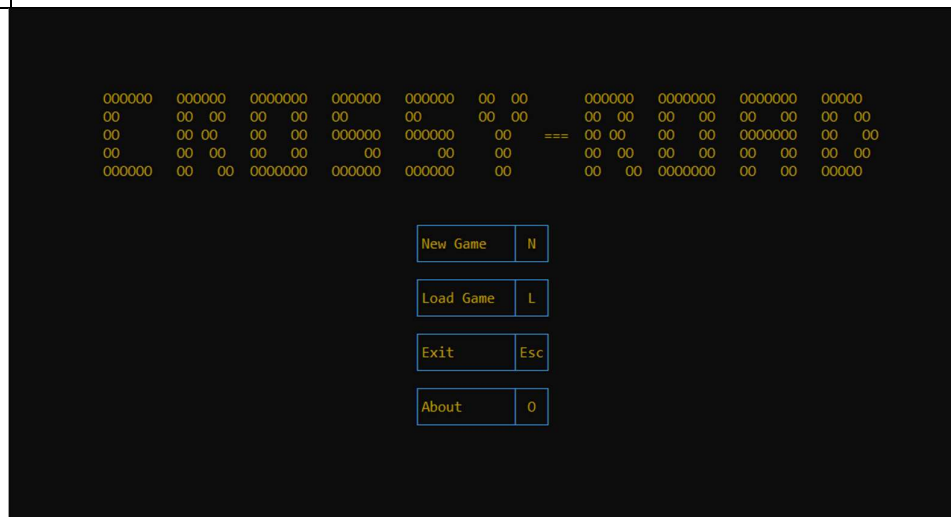
Đã được nhắc đến ở hàm processDead(), nhưng do đây là hàm trang trí nên được để cập sau. Hàm này được thiết kế in ra hiệu ứng sau khi hồn mà bay khỏi vùng chơi (playground) sau hàm processDead(), nó sẽ bay lên trời với nguyên lí cập nhật vị trí giống hàm moveCars() (dòng 27 đến dòng 35) và gặp Đức Phật (dòng 3 đến dòng 21). Sau đó chương trình sẽ hỏi bạn thao tác tiếp theo là chơi lại (phím R, dòng 40 – 41) hoặc về màn hình chính (phím bất kì, dòng 42 – 43). Ngay sau đó là lệnh `state = 0` thông báo người chơi đã thua để thread tạm dừng hoạt động.



Minh hoạ hình ảnh sau khi người chơi thua cuộc

Bước 15: Xây dựng hàm vẽ ra giao diện màn hình chính (Yêu cầu đồ án – hàm Trang trí)

Dòng	
1	<code>void startMenu() {</code>
2	<code> setTextColor(6);</code>
3	<code> GotoXY(13, 5); cout << "000000 000000 0000000 000000 000000</code> <code>00 00 000000 0000000 0000000 00000";</code>
4	<code> GotoXY(13, 6); cout << "00 00 00 00 00 00 00 00 00 00</code> <code>00 00 00 00 00 00 00 00 00 00";</code>
5	<code> GotoXY(13, 7); cout << "00 00 00 00 000000 000000</code> <code>00 == 00 00 00 00 0000000 00 00";</code>
6	<code> GotoXY(13, 8); cout << "00 00 00 00 00 00 00 00 00 00</code> <code>00 00 00 00 00 00 00 00 00 00";</code>
7	<code> GotoXY(13, 9); cout << "000000 00 00 0000000 000000 000000</code> <code>00 00 00 0000000 00 00 00000";</code>
8	<code> for (int i = 0; i <= 3; ++i)</code>
9	<code> drawSelectBox(51, 12 + 3 * i, 16, 2);</code>
10	<code> setTextColor(6);</code>
11	<code> GotoXY(52, 13); cout << "New Game";</code>
12	<code> GotoXY(65, 13); cout << "N";</code>
13	<code> GotoXY(52, 16); cout << "Load Game";</code>
14	<code> GotoXY(65, 16); cout << "L";</code>
15	<code> GotoXY(52, 19); cout << "Exit";</code>
16	<code> GotoXY(64, 19); cout << "Esc";</code>
17	<code> GotoXY(52, 22); cout << "About";</code>
18	<code> GotoXY(65, 22); cout << "O";</code>
19	<code> setTextColor(7);</code>
20	<code> GotoXY(0, 0);</code>
21	<code>}</code>



Minh hoạ giao diện màn hình chính

Phần 3: Xây dựng thư viện chứa hàm xử lý các thao tác được yêu cầu từ người chơi (gameCommand.h).

Bước 1: Xây dựng hàm bắt đầu trò chơi mới

Dòng	
1	<code>void startGame() {</code>
2	<code> system("cls");</code>
3	<code> resetData();</code>
4	<code> drawGameMenu();</code>
5	<code> drawBox(0, 0, WIDTH_CONSOLE, PLAYGROUND_SECTION_HEIGHT);</code>
6	<code> drawBox(0, PLAYGROUND_SECTION_HEIGHT + 1, WIDTH_CONSOLE,</code> <code> INFO_SECTION_HEIGHT);</code>
7	<code> state = 1;</code>

Hàm `startGame()` để khởi tạo lại dữ liệu ban đầu và tạo màn chơi mới cho người chơi, hàm này được gọi khi người chơi nhấn phím 'N'. Dòng 2 để xóa trắng màn hình, dòng 3 là lời gọi hàm `resetData()` nhằm khôi phục dữ liệu mặc định ban đầu. Kế đó là gọi hàm để vẽ các khung viền xung quanh. Dòng lệnh cuối cùng là quan trọng, khi `state = 1` thì các đối tượng mới chính thức được vẽ ra màn hình.

Bước 2: Xây dựng hàm thoát khỏi trò chơi, hàm tạm dừng và tiếp tục trò chơi

Dòng	
1	<code>void exitGame(HANDLE t) {</code>
2	<code> system("cls");</code>
3	<code> TerminateThread(t, 0);</code>
4	<code> deleteCarData();</code>
5	<code> exit(1);</code>
6	<code>}</code>
7	<code>void pauseGame(HANDLE t) {</code>
8	<code> SuspendThread(t);</code>
9	<code>}</code>
10	<code>void resumeGame(HANDLE t) {</code>
11	<code> ResumeThread(t);</code>
12	<code>}</code>

Trong hàm `exitGame()`, ta thực hiện xóa trắng màn hình và ngắt tiểu trình đang chạy, ta cần có `HANDLE` của tiểu trình cần ngắt. Do trong ứng dụng có sử dụng biến con trỏ nên khi thoát ta cần dọn dẹp các tài nguyên này bằng dòng 4.

Tương tự cho hàm `pauseGame()` và `resumeGame()` vẫn cần `HANDLE` của tiểu trình để tạm ngưng và tiếp tục chạy trò chơi.

Bước 3: Xây dựng hàm ghi và lưu lại dữ liệu người chơi vào file .txt

Dòng	
1	<code>void saveGame(string player_name, POINT player_pos) {</code>
2	<code> string fileName = player_name + ".txt";</code>
3	<code> fstream fileOutput;</code>
4	<code> fileOutput.open(fileName, ios::out ios::trunc); // erase exist</code>
5	<code> fileOutput << speed << "\n" << step << "\n";</code>
6	<code> for (int i = 0; i < MAX_CAR; i++) {</code>
7	<code> fileOutput << carInfo[i].length << "\n" <<</code> <code> carInfo[i].direction << "\n";</code>
8	<code> for (int j = 0; j < carInfo[i].length; j++)</code>
9	<code> fileOutput << carArray[i][j] << "\n";</code>
10	<code> }</code>
11	<code> for (int i = 0; i < speed; ++i) {</code>
12	<code> fileOutput << prevPos[i] << "\n";</code>
13	<code> }</code>
14	<code> fileOutput << player_pos.x << "\n" << player_pos.y;</code>
15	<code> fileOutput.close();</code>
16	<code>}</code>

Hàm `saveGame()` có chức năng cơ bản là ghi file sử dụng thư viện `fstream.h` để giúp người chơi có thể lưu lại quá trình chơi của mình để tiếp tục sau. Dòng 4 là để tạo và ghi file, trong trường hợp file đã tồn tại tại, lệnh *trunc* để xóa hết dữ liệu trong file đấy. Các thông số được ghi lên file gồm level và số bước đã di chuyển hiện tại, hướng di chuyển, độ dài và vị trí của từng xe, vị trí của các nhân vật đã về đích trước đó, vị trí người chơi hiện tại (dòng 5 đến 14). Toàn bộ dữ liệu trên được lưu vào file có cú pháp `filename.txt` (dòng 1, dòng 4) với `filename` do người chơi nhập từ bàn phím (không phân biệt chữ hoa chữ thường) từ hàm `main`.

Bước 4: Xây dựng hàm đọc và tải lại dữ liệu người chơi từ file .txt

Dòng	
1	<code>void loadGame(string player_name, POINT& player_pos) {</code>
2	<code> system("cls");</code>
3	<code> drawBox(0, 0, WIDTH_CONSOLE, PLAYGROUND_SECTION_HEIGHT);</code>
4	<code> drawBox(0, PLAYGROUND_SECTION_HEIGHT + 1, WIDTH_CONSOLE,</code> <code> INFO_SECTION_HEIGHT);</code>

5	<code>string fileName = player_name + ".txt";</code>
6	<code>fstream fileInput(fileName);</code>
7	<code>fileInput >> speed >> step;</code>
8	<code>for (int i = 0; i < MAX_CAR; i++) {</code>
9	<code>fileInput >> carInfo[i].length >> carInfo[i].direction;</code>
10	<code>carArray[i] = new int[carInfo[i].length];</code>
11	<code>for (int j = 0; j < carInfo[i].length; j++)</code>
12	<code>fileInput >> carArray[i][j];</code>
13	<code>}</code>
14	<code>setTextColor(6);</code>
15	<code>for (int i = 0; i < speed; ++i) {</code>
16	<code>fileInput >> prevPos[i];</code>
17	<code>drawCharacter({ prevPos[i], 2 }, "Y");</code>
18	<code>}</code>
19	<code>fileInput >> player_pos.x >> player_pos.y;</code>
20	<code>fileInput.close();</code>
21	<code>drawCharacter(player_pos, "Y");</code>
22	<code>drawCars();</code>
23	<code>drawInfo();</code>
24	<code>drawGameMenu();</code>
25	<code>}</code>

Hàm `loadGame()` nhờ vào thư viện `fstream` có khả năng đọc dữ liệu từ file `.txt`. Trước tiên, hàm thực hiện xóa trắng màn hình và vẽ lại khung viền trò chơi (dòng 2, 3, 4). Sau đó, hàm đọc các thông số lần lượt theo thứ tự đã lưu ở hàm `saveGame()`. Có một điểm cần lưu ý tại dòng 10 là ta phải cấp phát lại kích thước vùng nhớ mới cho mảng `carArray` tránh trường hợp thiếu hoặc thừa vùng nhớ (do `carArray` có thể đã được tạo ra trước đó từ hàm `resetData()`). Sau đó ta in các thông tin đã đọc được ra màn hình console (dòng 17, 22, 23, 24, 25). Lưu ý, hàm `loadGame()` chỉ tải và in dữ liệu ra màn hình console chứ không có chức năng vận hành, tạo chuyển động xe, nhân vật, do đó, sau khi hàm `loadGame()` chạy xong, mọi thứ “tạm thời đứng yên”.

Phần 4: Xây dựng thư viện chứa các hàm hỗ trợ di chuyển nhân vật trong game (gameMovement.h).

Tại thư viện này, ta xây dựng các hàm hỗ trợ nhân vật di chuyển qua trái, phải, lên, xuống. Chia làm hai phần chính là trái-phải và lên-xuống. Nhìn chung, khi nhận lệnh bàn phím người chơi, ta kiểm tra lệnh đấy có thể thực hiện không (có thể bước tới vị trí mới không), sau đó ta xoá nhân vật tại vị trí hiện tại (dòng 3, 11, 20, 28), cập nhật vị trí mới cho nhân vật (dòng 6, 14, 21, 29) và in nhân vật tại vị trí mới bằng hàm drawCharacter() (dòng 7, 15, 22, 30).

Đối với hai hàm chuyển động trái-phải thì không có vị trí nào là không thể qua trái hoặc qua phải được, vì nếu nhân vật chạm biên (khung viền) của vùng chơi và tiếp tục di chuyển, nhân vật sẽ cập nhật vị trí mới là tại biên của phía đối diện. Đối với hàm chuyển động lên xuống, nhân vật không được đi xuống nếu đang đứng ở vị trí xuất phát, cũng như không thể đi lên nếu đang ở vị trí đích.

Dòng	
1	<code>void moveRight() {</code>
2	<code> step++;</code>
3	<code> drawCharacter(player_pos, " ");</code>
4	<code> if (player_pos.x > WIDTH_CONSOLE - 2)</code>
5	<code> player_pos.x = 0;</code>
6	<code> player_pos.x++;</code>
7	<code> drawCharacter(player_pos, "Y");</code>
8	<code>}</code>
9	<code>void moveLeft() {</code>
10	<code> step++;</code>
11	<code> drawCharacter(player_pos, " ");</code>
12	<code> if (player_pos.x < 2)</code>
13	<code> player_pos.x = WIDTH_CONSOLE;</code>
14	<code> player_pos.x--;</code>
15	<code> drawCharacter(player_pos, "Y");</code>
16	<code>}</code>
17	<code>void moveDown() {</code>
18	<code> if (player_pos.y < PLAYGROUND_SECTION_HEIGHT) {</code>
19	<code> step++;</code>
20	<code> drawCharacter(player_pos, " ");</code>
21	<code> player_pos.y++;</code>

22	drawCharacter(player_pos, "Y");
23	}
24	}
25	void moveUp() {
26	if (player_pos.y > 2) {
27	step++;
28	drawCharacter(player_pos, " ");
29	player_pos.y--;
30	drawCharacter(player_pos, "Y");
31	}
32	}

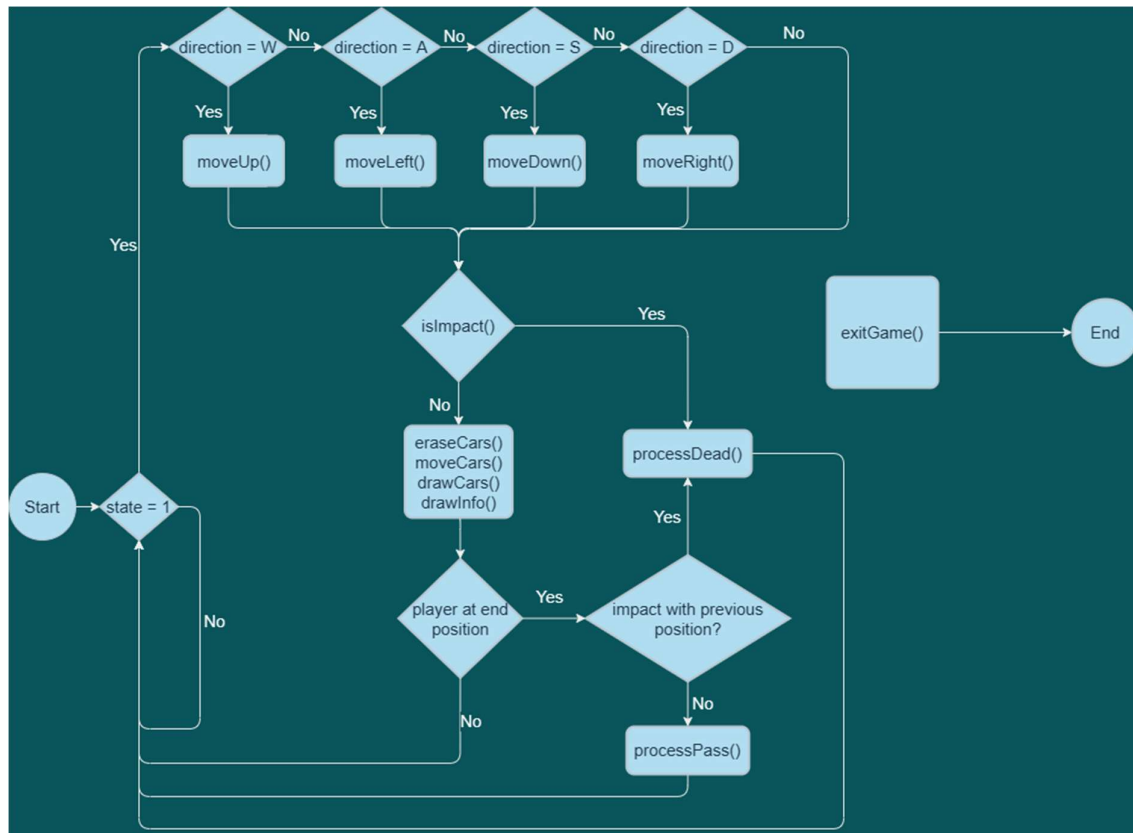
Phần 5: Xây dựng thư viện chứa hàm chạy cho tiểu trình, quản lý những sự kiện diễn ra trên màn hình console (consoleControl.h).

Xây dựng hàm subThread() quản lý những chuyển động của nhân vật, xe, mọi sự kiện có thể diễn ra trong game như: thua cuộc, qua màn, di chuyển,...

Dòng	
1	<code>void subThread() {</code>
2	<code> int delay = 0;</code>
3	<code> int time2stop = speed * 10 + 1;</code>
4	<code> while (1) {</code>
5	<code> if (state) {</code>
6	<code> switch (direction) {</code>
7	<code> case 'A': moveLeft(); break;</code>
8	<code> case 'D': moveRight(); break;</code>
9	<code> case 'W': moveUp(); break;</code>
10	<code> case 'S': moveDown(); break;</code>
11	<code> }</code>
12	<code> direction = ' ';</code>
13	<code> if (isImpact(player_pos)) {</code>
14	<code> processDead();</code>
15	<code> delay = 1;</code>
16	<code> time2stop = 0;</code>
17	<code> }</code>
18	<code> delay == 1 ? time2stop++ : time2stop--;</code>
19	<code> if (time2stop == 0)</code>
20	<code> delay = 1;</code>
21	<code> else if (time2stop == speed * 10 + 1)</code>
22	<code> delay = 0;</code>
23	<code> if (delay == 0) {</code>
24	<code> eraseCars();</code>
25	<code> moveCars();</code>
26	<code> drawCars();</code>
27	<code> }</code>
28	<code> drawInfo();</code>
29	<code> if (player_pos.y == 2) {</code>
30	<code> if (player_pos.x == prevPos[0] player_pos.x</code>
31	<code> == prevPos[1]) {</code>
	<code> processDead();</code>

32	delay = 1;
33	time2stop = 0;
34	}
35	else {
36	processPass(player_pos);
37	delay = 0;
38	time2stop = speed * 10 + 1;
39	}
40	}
41	Sleep(time2stop);
42	}
43	}
44	}

Hàm *subThread* có thể được minh họa qua flowchart sau:



Có thể thấy hàm *subThread()* là một vòng lặp vô hạn, nhưng những dòng lệnh chỉ được thực thi khi *state* = true tức nhân vật vẫn còn sống. Nếu thỏa điều kiện này, hàm sẽ kiểm tra có lệnh nhập từ bàn phím người chơi không để thực thi hàm di chuyển. Sau đó, kiểm tra có va chạm sau di chuyển không rồi mới bắt đầu cho di chuyển các xe theo hướng định sẵn (tức cập nhật vị trí cho các xe),

cuối cùng là kiểm tra nếu người chơi đã đưa nhân vật về đích thì có va chạm với các nhân vật trước đó không. Sau khi hoàn thành các bước xử lý trên thì vòng lặp lại tiếp tục.

Như đã nói tại phần yêu cầu đồ án, ta sử dụng biến *delay* để thông báo khi nào thì sẽ cập nhật và di chuyển các xe trên màn hình console (*delay* = 0). Khi biến *delay* = 1, các hàm từ dòng từ 24 đến 26 sẽ không được gọi, khiến cho tất cả các xe dừng nhưng người chơi vẫn di chuyển nhân vật được.

Ta hoàn toàn có thể cập nhật lại giá trị của biến *delay* để có sự xen kẽ chạy và dừng, nhưng do tốc độ xử lý của máy tính thì khoảng thời gian xen kẽ ấy không đáng kể và không tạo sự khác biệt. Vì thế ta thiết lập một biến *time2stop* coi nó như một chiếc đồng hồ đếm ngược để chương trình biết khi nào nên cập nhật biến *delay*.

Thiết lập này có tính chất: khi *delay* = 0 thì *time2stop min* (= 0) và khi *delay* = 1 thì *time2stop max* (= *x*). Tại dòng 18, khi *delay* = 0, biến *time2stop* giảm dần và ngược lại. Khi đó, khoảng thời gian xen kẽ giữa chạy và dừng là khoảng thời gian vòng lặp chạy *x* lần, bổ sung thêm hàm Sleep() được truyền vào tham số ngẫu nhiên (ở đây ta chọn luôn *time2stop*) ở dòng 41 sẽ tạo được sự khác nhau rõ ràng giữa hai nhịp dừng chạy, lưu ý nên để tham số tại hàm Sleep() này đủ nhỏ để không ảnh hưởng đến sự mượt mà của chuyển động nhân vật (khi này hoàn toàn có thể di chuyển).

Phần 6: Xây dựng hàm main và file mã nguồn, sử dụng các hàm và thư viện đã xây dựng trước đó (CrossyRoadProject.cpp)

Đây là hàm phức tạp nhất trong toàn bộ đồ án vì nó dùng để quản lý mọi nguồn tài nguyên, dữ liệu, sự kiện diễn ra trong suốt quá trình từ khi game được khởi động đến khi được đóng. Hàm main thuộc main thread – chạy song song với tiểu trình (sub thread) mà chúng ta tạo ra, từ đây sẽ được hiểu hàm main chạy song song với hàm subThread() đã được tạo dựng ở trên.

Dòng	
1	<code>int main() {</code>
2	<code> int temp;</code>
3	<code> FixConsoleWindow();</code>
4	<code> thread t1(subThread);</code>
5	<code> while (1) {</code>
6	<code> system("cls");</code>
7	<code> startMenu();</code>
8	<code> while (1) {</code>
9	<code> temp = toupper(_getch());</code>
10	<code> if (temp == 'L') {</code>
11	<code> resetData();</code>
12	<code> fillBox(0, PLAYGROUND_SECTION_HEIGHT + 2,</code>
13	<code> WIDTH_CONSOLE, 0, " ");</code>
14	<code> GotoXY(5, PLAYGROUND_SECTION_HEIGHT + 2);</code>
15	<code> setTextColor(3);</code>
16	<code> cout << "Enter your name to continue playing: ";</code>
17	<code> while (1) {</code>
18	<code> setTextColor(6);</code>
19	<code> getline(cin, player_name);</code>
20	<code> if (!isFileExist(player_name)) {</code>
21	<code> fillBox(0,</code>
22	<code> PLAYGROUND_SECTION_HEIGHT + 2, WIDTH_CONSOLE, 0, " ");</code>
23	<code> GotoXY(5, PLAYGROUND_SECTION_HEIGHT</code>
24	<code> + 2);</code>
25	<code> setTextColor(3);</code>
26	<code> cout << "Can't find ";</code>
27	<code> setTextColor(6);</code>
	<code> cout << player_name << "'s";</code>
	<code> setTextColor(3);</code>
	<code> cout << " data.Please re-enter: ";</code>

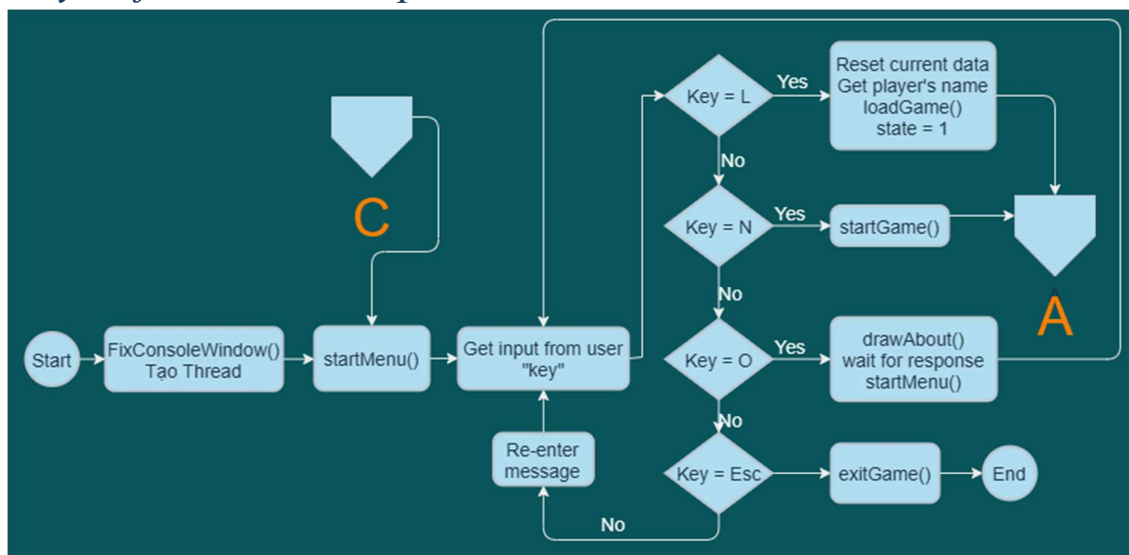
28	}
29	else
30	break;
31	}
32	loadGame(player_name, player_pos);
33	state = 1;
34	break;
35	}
36	else if (temp == 'N') {
37	startGame();
38	break;
39	}
40	else if (temp == 27) {
41	exitGame(t1.native_handle());
42	return 0;
43	}
44	else if (temp == 'O') {
45	system("cls");
46	drawAbout();
47	temp = _getch();
48	system("cls");
49	startMenu();
50	}
51	else {
52	GotoXY(5, PLAYGROUND_SECTION_HEIGHT + 2);
53	setTextColor(6);
54	cout << "It's not an option. Please choose
55	again.";
56	}
57	}
58	while (1) {
59	temp = toupper(_getch());
60	if (state) {
61	if (temp == 27) {
62	state = 0;
63	Sleep(40);
64	break;
65	}
66	else if (temp == 'P')
67	pauseGame(t1.native_handle());
68	else if (temp == 'R')
69	resumeGame(t1.native_handle());
70	else if (temp == 'K') {

69	state = 0;
70	Sleep(60);
71	system("cls");
72	if (player_name == DEFAULT_PLAYER_NAME) {
73	GotoXY(35, 12);
74	setTextColor(3);
75	cout << "Enter your name : ";
76	setTextColor(6);
77	getline(cin, player_name);
78	}
79	else {
80	GotoXY(28, 12);
81	setTextColor(3);
82	cout << "Enter your new name or press Enter to save to the file ";
83	setTextColor(6);
84	cout << player_name << ".txt";
85	string new_name;
86	setTextColor(3);
87	cout << ": ";
88	setTextColor(6);
89	getline(cin, new_name);
90	if (new_name != "")
91	player_name = new_name;
92	}
93	saveGame(player_name, player_pos);
94	GotoXY(35, 13);
95	setTextColor(3);
96	cout << "Your game have been saved to the file ";
97	setTextColor(6);
98	cout << player_name << ".txt";
99	setTextColor(7);
100	break;
101	}
102	else if (temp == 'D' temp == 'A' temp == 'W' temp == 'S') {
103	resumeGame(t1.native_handle());
104	direction = temp;
105	}
106	Else
107	continue;
108	}

109	<code>else {</code>
110	<code>if (temp == 'R') {</code>
111	<code>startGame();</code>
112	<code>}</code>
113	<code>else {</code>
114	<code>system("cls");</code>
115	<code>break;</code>
116	<code>}</code>
117	<code>}</code>
118	<code>}</code>
119	<code>}</code>
120	<code>}</code>

Hàm main() được chia làm 2 phần chính. Phần 1 là phần tiền xử lý tại màn hình chính. Tại phần này người chơi có thể bắt đầu trò chơi mới, tải lại phần chơi đã lưu, xem thông tin nhóm sáng tạo đồ án và thoát khỏi trò chơi.

Đây là flowchart của phần 1

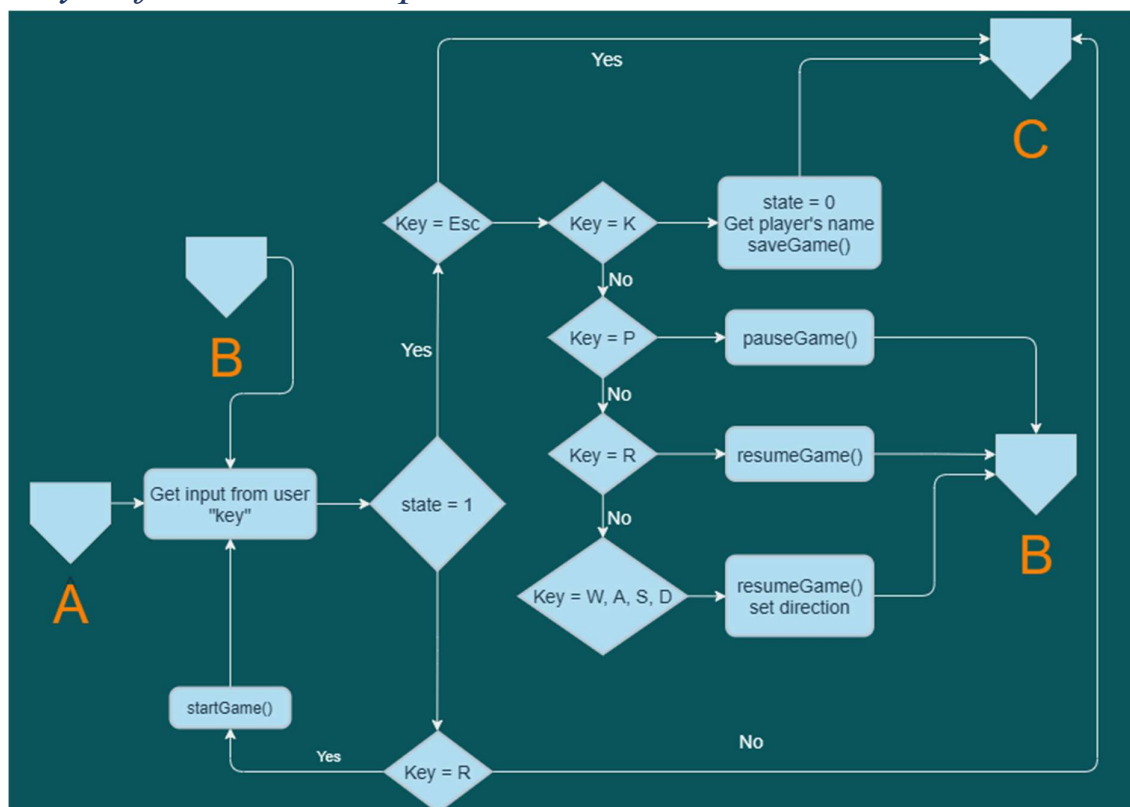


Nhìn vào hình minh họa, hàm main() đầu tiên sẽ in ra giao diện menu bằng hàm startMenu(), sau đó đợi người chơi nhập sự lựa chọn. Nếu người chơi bắt đầu game mới, hàm main() sẽ gọi hàm startGame() đã được xây dựng. Nếu tải lại phần chơi cũ, một thao tác được bổ sung để chương trình chạy an toàn hơn đó là kiểm tra xem file người chơi muốn tải lại có tồn tại hay không, nếu file đấy không tồn tại thì yêu cầu người chơi nhập lại (dòng 19 đến dòng 28). Sau đó, chương trình sẽ đặt lại mọi thông số về mặc định, chú ý có phần *state = 1*, chỉ khi có dòng này thì các xe được in từ phần loadGame() mới bắt đầu chuyển động. Nếu người chọn 1 trong 2 lựa chọn trên, chương trình sẽ tiếp tục thực thi phần chơi game (phần 2 hay còn gọi là phần A trên flowchart). Nếu

người chơi muốn xem thông tin nhóm sáng lập, chương trình gọi hàm `drawAbout()`, đợi người dùng phản hồi (nhấn phím bất kì, dòng 47) sẽ gọi hàm `startMenu()` để xóa màn hình và in lại menu sau đó dùng vòng lặp quay lại bước đợi người chơi nhập sự lựa chọn. Còn với trường hợp người nhấn phím `Esc`, chương trình sẽ thoát (kết thúc chương trình). Nếu người chơi nhập sự lựa chọn không có trong menu, chương trình sẽ hiện thông báo nhắc nhở người chơi nhập lại.

Phần 2 (phần A) chính là phần xử lý trong quá trình chơi game, liên hệ trực tiếp và chặt chẽ với hàm subThread().

Đây là flowchart của phần 2



Phần A sau khi kết thúc các bước thực thi sẽ cho ra hai trường hợp kết quả: phần B (trò chơi tiếp tục, đợi người chơi nhập sự lựa chọn) và phần C (trò chơi kết thúc, trở về màn hình chính). Sau khi người dùng nhập sự lựa chọn, chương trình sẽ kiểm tra tình trạng sống/chết của nhân vật, sau đó lựa chọn các trường hợp đúng để thực thi chương trình.

Trong trường hợp nhân vật còn sống, người chơi có thể tạm dừng game bằng hàm `pauseGame()`, tiếp tục chơi bằng hàm `resumeGame()` tương ứng với các phím chức năng. Ở phần di chuyển nhân vật (Key = W, A, S, D) chương trình sẽ cập nhật hướng di chuyển của nhân vật, hàm `subThread()` đang chạy vô

hạn song song sẽ nhận được lệnh và di chuyển nhân vật theo yêu cầu người chơi. Ba trường hợp nêu trên sẽ dẫn đến kết quả là phần B. Ngoài ra, ở phần lưu game (Key = K) chương trình cập nhật trạng thái *state* = 0 để hàm *subThread()* ngưng cập nhật vị trí xe và không cho phép nhân vật di chuyển nữa, dòng 70 sử dụng hàm *Sleep()* với tác dụng tạo khoảng thời gian để *subThread()* cập nhật giá trị của biến *state*. Nếu không có lệnh này, trong một số trường hợp xấu là khi cập nhật biến *state* thì *subThread()* chạy song song vẫn chưa đi tới lệnh cập nhật vị trí và in xe, nếu ta in ngay các hiệu ứng mới thì dễ dẫn đến các xe và hiệu ứng mới này ghi đè lên nhau, vì thế cần có hàm *Sleep()* để *subThread()* kết thúc vòng lặp hiện tại và cập nhật giá trị biến *state* = 0 mới. Một thao tác bổ sung cho quá trình lưu trò chơi là nếu người chơi đang chơi phần chơi được tải lại và tiếp tục lưu, chương trình hỏi người dùng có muốn lưu đè vào file cũ hay lưu vào file mới (dòng 79 đến dòng 92). Nếu người chơi muốn thoát ra để trở về màn hình chính, ta cũng cho tạm dừng thread như cách đã làm với trường hợp trên, đặt *state* = 0 và *Sleep()* trong khoảng thời gian nhất định. Hai trường hợp tiếp theo này sẽ dẫn đến kết quả là phần C.

Trong trường hợp trạng thái nhân vật là đã chết (thua), khi này hàm *subThread()* đã thực thi lệnh *processDead()* và in ra giao diện thua cuộc (Đức Phật và Hồn Ma). Người chơi sẽ được hỏi xem tiếp tục chơi hay không? Nếu người chơi chọn phím 'R', chương trình gọi hàm *startGame()* để bắt đầu phần chơi mới mà không cần trở về màn hình chính, trường hợp này cũng sẽ dẫn đến kết quả là phần B). Nếu người chơi chọn phím bất kỳ, chương trình sẽ đưa người chơi về màn hình chính hay kết quả là phần C.

Khi chương trình xác định kết quả là phần B, nhờ vào vòng lặp vô hạn của phần 2, trò chơi tiếp tục. Nếu kết quả là phần C, nhờ vào vòng lặp vô hạn bao quanh phần 1 và phần 2, ta dễ dàng quay trở lại phần 1 và thực hiện các thao tác từ đầu.

NGUỒN THAM KHẢO

- ❖ Tài liệu hướng dẫn thực hiện đồ án bằng qua đường do giáo viên Trương Toàn Thịnh cung cấp.
- ❖ cplusplus: <https://www.cplusplus.com/>
- ❖ cppreference: <https://en.cppreference.com/w/>
- ❖ Stackoverflow: <https://stackoverflow.com/>
- ❖ Geeksforgeeks: <http://geeksforgeeks.org>
- ❖ Dạy nhau học: <https://daynhauhoc.com/>
- ❖ Source code của user GitHub toanmadrid:
<https://gist.github.com/toanmadrid/07aac3102973dd4d6c5cd2ea27c6dd15>
- ❖ Ascii Art:
 - <https://www.asciiart.eu/religion/buddhism>
 - <https://www.asciiart.eu/mythology/ghosts>