HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN
HỌC PHẦN: Ngôn ngữ lập trình C++
MÃ HỌC PHẦN: INT1339

ĐỀ TÀI: Thiết kế và cài đặt trò chơi "Car Racing bằng SDL3 và SDL_ttf

	Các sinh viên thực hiện		Mã Sinh Viên
1	***Trần Phi Anh***	:	B24DCCN047
2	***Mai Nhật Minh***	:	B24DCCN391
3	***Bùi Hà Hải Đạt***	:	B24DCCN102

Tên nhóm:12-3

Tên lớp: D24-056

Giảng viên hướng dẫn: Ninh Thị Thu Trang

HÀ NỘI 2025

PHÂN CÔNG NHIỆM VỤ NHÓM THỰC HIỆN

TT	Công việc / Nhiệm vụ	SV thực hiện	Thời hạn hoàn thành
1	Viết hàm xử lí chính	Trần Phi Anh	15/10//2025
2	Xây dựng đối tượng + Viết báo cáo	Mai Nhật Minh	18/10/2025
3	Xử lý file, xử lí lỗi + Tìm hiểu thư viện	Bùi Hà Hải Đạt	15/10/2025

NHÓM THỰC HIỆN TỰ ĐÁNH GIÁ

TT	SV thực hiện	Thái độ tham gia	Mức hoàn thành CV	Kỹ năng giao tiếp	Kỹ năng hợp tác	Kỹ năng lãnh đạo
1	Trần Phi Anh	5	5	5	5	5
2	Mai Nhật Minh	5	5	5	5	5
3	Bùi Hà Hải Đạt	5	5	5	5	5

MỤC LỤC

MỞ ĐẦU

Trò chơi Car Racing là một mini-game lái xe tránh chướng ngại vật được phát triển bằng thư viện SDL3 và SDL_ttf cho việc hiển thị văn bản. Mục tiêu của báo cáo này là mô tả thiết kế, cài đặt, thử nghiệm và đề xuất hướng phát triển cho trò chơi, đồng thời trình bày mã nguồn chính và giải thích các chức năng quan trọng.

Mục tiêu của dự án:

Thiết kế một trò chơi đơn giản, dễ hiểu để minh họa lập trình đồ họa 2D với SDL3.

Thực hành tổ chức mã, xử lý sự kiện, vẽ đồ họa 2D, quản lý va chạm và hiển thị văn bản.

Đạt các CLO liên quan: triển khai hàm xử lý, sử dụng cấu trúc dữ liệu, xử lý I/O cơ bản và ngoại lệ.

Phạm vi: trò chơi chạy trên cửa sổ 800x600, người chơi điều khiển xe đi ngang để tránh các xe địch lao xuống từ trên. Có menu chính, màn hướng dẫn, gameplay và màn game over.

CHƯƠNG 1. TỔNG QUAN VỀ GAME

1.1 Giới thiệu

Trò chơi **"Car Racing"** là một mini-game 2D mô phỏng lái xe tránh chướng ngại vật phát triển bằng thư viện **SDL3** và **SDL_ttf**. Người chơi điều khiển một chiếc xe nhìn từ trên xuống, di chuyển sang trái/phải để né các xe địch rơi xuống từ phía trên màn hình. Trò chơi nhằm mục đích minh họa các kỹ thuật lập trình đồ họa 2D cơ bản: khởi tạo cửa sổ và renderer SDL, vẽ hình học cơ bản, xử lý sự kiện bàn phím, cập nhật trạng thái theo khung hình, kiểm tra va chạm và hiển thị văn bản.

1.2 Các thành phần và luật chơi

1.2.1 Các đối tượng chính trong game

Player (Xe người chơi): đối tượng do người chơi điều khiển bằng phím A (trái) và D (phải). Kích thước: 40 × 80 px.

OpponentCar (Xe địch): các xe xuất hiện từ phía trên màn hình, chạy xuống dưới theo trục Y với tốc độ có thể thay đổi. Mỗi xe có thuộc tính posX, posY, currentSpeed, isActive, carColor.

Road (Đường đua): vùng di chuyển chính ở giữa cửa sổ, có width = 500 px, nằm chính giữa.

UI: hiển thị điểm số (text), thanh điểm (bar) ở hai bên đường, menu chính, hướng dẫn và màn hình game over.

1.2.2 Luật chơi

Người chơi điều khiển xe để tránh chạm vào các xe địch. Mỗi xe địch đi ra khỏi đáy màn hình sẽ được respawn lên đầu và người chơi được **+1 điểm**.

Tốc độ xe địch tăng dần theo điểm: currentSpeed = ENEMY_INITIAL_SPEED + score * 0.1.

Ở các mốc điểm (ví dụ: 5, 15) sẽ kích hoạt thêm xe địch để tăng độ khó.

Va chạm giữa bounding box của người chơi và bất kỳ xe địch nào => **Game Over**.

1.3 Kết chương

Chương 1 trình bày mục tiêu, cấu trúc tổng quan và luật chơi cơ bản của trò chơi. Các chương sau sẽ đi sâu vào thiết kế cài đặt (Chương 2), và kiểm thử + đánh giá (Chương 3).

CHƯƠNG 2. THIẾT KẾ VÀ CÀI ĐẶT CÁC THÀNH PHẦN GAME

2.1 Các hàm xử lý chính (CLO1)

2.1.1 Luồng tổng quát chương trình

main()

```
502 int main(int argc, char* argv[]) {
503     (void)argc;
504     (void)argv;
505 
506     if (!initializeSDL()) {
507         cerr << "Application failed to initialize!" << endl;
508         return 1;
509     }
510 
511     GameManager gameManager;
512 
513     while (gameManager.isRunning()) {
514         gameManager.handleEvents();
515         gameManager.updateLogic();
516         gameManager.renderGame();
517 
518         SDL_Delay(1000 / 60);
519     }
520 
521     cleanupSDL();
522     return 0;
523 }
```

Gọi initializeSDL() để khởi tạo SDL, tạo Window/Renderer và load font.

Tạo GameManager gameManager;

Vào vòng lặp chính:

```
while (gameManager.isRunning()) {
    gameManager.handleEvents();
    gameManager.updateLogic();
    gameManager.renderGame();
    SDL_Delay(1000/60);  (cố gắng giữ ~60 FPS)
}
```

Gọi cleanupSDL() để giải phóng tài nguyên trước khi thoát.

2.1.2 Hàm khởi tạo và dọn dẹp

bool initializeSDL()

```
450  bool initializeSDL() {
451      if (SDL_Init(SDL_INIT_VIDEO) < 0) {
452          cerr << "SDL failed to initialize: " << SDL_GetError() << endl;
453          return false;
454      }
455
456      g_pWindow = SDL_CreateWindow("GROUP 3: The Race", WINDOW_W, WINDOW_H, 0);
457      if (g_pWindow == nullptr) {
458          cerr << "Window creation failed: " << SDL_GetError() << endl;
459          return false;
460      }
461
462      g_pRenderer = SDL_CreateRenderer(g_pWindow, nullptr);
463      if (g_pRenderer == nullptr) {
464          cerr << "Renderer creation failed: " << SDL_GetError() << endl;
465          return false;
466      }
467
468      if (TTF_Init() == -1) {
469          cerr << "SDL_ttf failed to initialize: " << SDL_GetError() << endl;
470          return false;
471      }
472
473      try {
474          g_pFont = TTF_OpenFont("arial.ttf", 20);
475          if (g_pFont == nullptr) {
476              throw runtime_error("Không thể load font: arial.ttf. Font phải nằm trong thư mục chạy.");
477          }
478      } catch (const runtime_error& e) {
479          cerr << "LỖI FONT: " << e.what() << " " << SDL_GetError() << endl;
480          return false;
481      }
482
483      srand(time(0));
484      return true;
```

Dòng 451: Gọi hàm **SDL_Init()** với cờ SDL_INIT_VIDEO để **khởi tạo hệ thống video** của SDL (cần thiết để tạo cửa sổ và hiển thị đồ họa).

- Hàm trả về **0 nếu thành công**, và một giá trị âm nếu thất bại. Điều kiện SDL_Init(...) < 0 kiểm tra xem việc khởi tạo có thất bại không.

Dòng 452: Nếu thất bại, in thông báo lỗi ra luồng lỗi chuẩn (cerr). Hàm SDL_GetError() trả về chuỗi mô tả lỗi cuối cùng mà SDL gặp phải.

Dòng 453: Trả về false để báo hiệu rằng việc khởi tạo đã thất bại.

Dòng 456: Gọi hàm **SDL_CreateWindow()** để tạo một cửa sổ mới.

- "GROUP 3: The Race": Tiêu đề của cửa sổ.
- WINDOW_W, WINDOW_H: Các hằng số (được định nghĩa ở đâu đó khác) xác định chiều rộng và chiều cao của cửa sổ.
- 0: Các cờ (flags) cho cửa sổ (ở đây là không có cờ đặc biệt nào).
- Con trỏ đến cửa sổ được lưu vào biến toàn cục g_pWindow.

Dòng 457: Kiểm tra xem g_pWindow có phải là nullptr (con trỏ rỗng) hay không. nullptr có nghĩa là cửa sổ không thể được tạo.

Dòng 458-459: Nếu thất bại, in thông báo lỗi và trả về false.

Dòng 461: Gọi hàm SDL_CreateRenderer() để **tạo một bộ render** (công cụ vẽ) liên kết với cửa sổ g_pWindow.

- Bộ render là thành phần quan trọng để vẽ đồ họa (hình ảnh, hình học, văn bản) lên cửa sổ.
- nullptr: Đối số thứ hai (renderer flags) cho phép SDL tự chọn bộ render tốt nhất có sẵn.
- Con trỏ đến bộ render được lưu vào biến toàn cục g_pRenderer.

Dòng 462: Kiểm tra xem g_pRenderer có phải là nullptr (thất bại) hay không.

Dòng 463-464: Nếu thất bại, in thông báo lỗi và trả về false.

Dòng 472: Bắt đầu khối try, nơi đặt các lệnh có khả năng gây ra ngoại lệ (exception).

Dòng 473: Gọi hàm **TTF_OpenFont()** để **tải font** từ file "arial.ttf" với kích thước.

- Con trỏ đến font (nếu thành công) được lưu vào biến toàn cục g_pFont.

Dòng 474: Kiểm tra xem g_pFont có phải là nullptr (tải thất bại) hay không.

Dòng 475: Nếu font không tải được, **ném một ngoại lệ** runtime_error với thông báo tiếng Việt rõ ràng.

Dòng 478: Bắt đầu khối catch, nơi xử lý ngoại lệ runtime_error nếu nó bị ném ra trong khối try.

Dòng 479: In thông báo lỗi ra cerr.

- e.what(): Lấy chuỗi thông báo lỗi từ ngoại lệ (runtime_error) đã ném (ví dụ: "Không thể load font...").
- Sau đó, in thêm lỗi từ **SDL_GetError()** (đôi khi cung cấp thông tin chi tiết hơn về lỗi file).

Dòng 480: Trả về false nếu việc tải font thất bại.

Dòng 483: Gọi hàm srand() để **khởi tạo (seed)** cho bộ sinh số ngẫu nhiên giả (pseudorandom number generator) của C/C++.

- Đối số time(0) (thời gian hiện tại) thường được dùng để đảm bảo chuỗi số ngẫu nhiên là khác nhau mỗi lần chương trình chạy

void cleanupSDL()

```
487  void cleanupSDL() {
488      if (g_pFont != nullptr) {
489          TTF_CloseFont(g_pFont);
490          g_pFont = nullptr;
491      }
492      TTF_Quit();
493
494      SDL_DestroyRenderer(g_pRenderer);
495      SDL_DestroyWindow(g_pWindow);
496      g_pWindow = nullptr;
497      g_pRenderer = nullptr;
498      SDL_Quit();
499  }
500
```

Dòng 488: Kiểm tra xem con trỏ font toàn cục g_pFont có trỏ đến một đối tượng hợp lệ không (tức là font đã được tải thành công trước đó). Điều này giúp tránh lỗi khi cố gắng giải phóng một con trỏ rỗng.

Dòng 489: Nếu g_pFont hợp lệ, gọi hàm **TTF_CloseFont()** để **giải phóng bộ nhớ** đã được cấp phát cho font đó.

Dòng 490: Gán g_pFont về nullptr sau khi giải phóng. Đây là một thói quen lập trình tốt để đảm bảo con trỏ không còn trỏ đến vùng nhớ đã được giải phóng (gọi là dangling pointer).

Dòng 492: Gọi hàm **TTF_Quit()** để **hủy khởi tạo thư viện SDL_ttf**.

Dòng 494: Gọi hàm SDL_DestroyRenderer() để **hủy và giải phóng bộ nhớ** của bộ render đồ họa (g_pRenderer).

Dòng 495: Gọi hàm SDL_DestroyWindow() để **hủy và giải phóng bộ nhớ** của cửa sổ ứng dụng (g_pWindow).

- **Lưu ý quan trọng:** Cần phải hủy Renderer **trước** khi hủy Window, vì Renderer phụ thuộc vào Window.

Dòng 496-497: Gán các con trỏ toàn cục g_pWindow và g_pRenderer về nullptr để ngăn ngừa lỗi truy cập không hợp lệ sau này.

Dòng 498: Gọi hàm **SDL_Quit()** để **hủy khởi tạo toàn bộ thư viện SDL**. Hàm này sẽ giải phóng tất cả các hệ thống con (như video, âm thanh,...) đã được khởi tạo bởi SDL_Init().

Dòng 499: Kết thúc hàm **cleanupSDL()**.

2.1.3 Xử lý input và sự kiện

GameManager::handleEvents()

```
307  void handleEvents() {
308      SDL_Event event;
309      while (SDL_PollEvent(&event) != 0) {
310          if (event.type == SDL_EVENT_QUIT) {
311              m_isRunning = false;
312          } else if (event.type == SDL_EVENT_KEY_DOWN) {
313              if (m_currentScreen == SCREEN_RACING && event.key.key == SDLK_ESCAPE) {
314                  m_currentScreen = SCREEN_GAMEOVER;
315                  saveHighScore();
316              }
317          } else if (event.type == SDL_EVENT_MOUSE_BUTTON_DOWN) {
318              float mouseX, mouseY;
319              SDL_GetMouseState(&mouseX, &mouseY);
320
321              if (m_currentScreen == SCREEN_MAIN_MENU) {
322                  if (mouseX > WINDOW_W/2 - 100 && mouseX < WINDOW_W/2 + 100) {
323                      if (mouseY > 200 && mouseY < 250) {
324                          initRacing();
325                          m_currentScreen = SCREEN_RACING;
326                      } else if (mouseY > 270 && mouseY < 320) {
327                          m_currentScreen = SCREEN_INSTRUCTIONS;
328                      } else if (mouseY > 340 && mouseY < 390) {
329                          m_isRunning = false;
330                      }
331                  }
332              } else if (m_currentScreen == SCREEN_INSTRUCTIONS) {
333                  m_currentScreen = SCREEN_MAIN_MENU;
334              } else if (m_currentScreen == SCREEN_GAMEOVER) {
335                  m_currentScreen = SCREEN_MAIN_MENU;
336              }
337          }
338      }
339
340      if (m_currentScreen == SCREEN_RACING) {
341          m_player.handleInput();
342      }
343  }
344
```

Dòng 307: **void handleEvents()** {: Bắt đầu định nghĩa hàm xử lý sự kiện.

Dòng 308: **SDL_Event event;**: Khai báo biến **event** kiểu SDL_Event để lưu thông tin sự kiện được lấy ra.

Dòng 309: while (SDL_PollEvent(&event) != 0) {: **Vòng lặp Sự kiện:** Lấy và xử lý từng sự kiện (nhấn phím, nhấp chuột, v.v.) trong hàng đợi cho đến khi hết.

Dòng 310: if (event.type == SDL_QUIT) {: Kiểm tra nếu loại sự kiện là **đóng cửa sổ** (SDL_QUIT).

Dòng 311: m_isRunning = false;: Đặt cờ trạng thái chạy game thành false để **thoát game** ở cuối Game Loop.

Dòng 312: } else if (event.type == SDL_EVENT_KEY_DOWN) {: Kiểm tra nếu sự kiện là **nhấn một phím xuống**.

Dòng 313: if (m_currentScreen == SCREEN_RACING && event.key.keysym.sym == SDLK_ESCAPE) {: Kiểm tra kép: Game đang ở màn hình **Đua xe** VÀ phím nhấn là **ESC**.

Dòng 314: m_currentScreen = SCREEN_GAMEOVER;: Chuyển màn hình hiện tại sang **GAME OVER**.

Dòng 315: **saveHighScore();:** Gọi hàm lưu điểm cao của người chơi.

Dòng 317: } else if (event.type == SDL_EVENT_MOUSE_BUTTON_DOWN) {: Kiểm tra nếu sự kiện là **nhấn nút chuột** xuống.

Dòng 318: float mouseX, mouseY;: Khai báo biến để lưu tọa độ X, Y của chuột.

Dòng 319: SDL_GetMouseState(&mouseX, &mouseY);: Lấy tọa độ chuột hiện tại.

Dòng 321: if (m_currentScreen == SCREEN_MAIN_MENU) {: **Xử lý nhấp chuột trong Menu Chính.**

Dòng 322: if (mouseX > WINDOW_W / 2 - 100 && mouseX < WINDOW_W / 2 + 100) {: Kiểm tra tọa độ X của chuột có nằm trong vùng nút bấm hay không.

Dòng 323: if (mouseY > 200 && mouseY < 250) {: Kiểm tra tọa độ Y có nằm trong vùng nút **"START RACE"** không.

Dòng 324: initRacing();: Nếu đúng, gọi hàm khởi tạo cuộc đua mới.

Dòng 325: m_currentScreen = SCREEN_RACING;: Chuyển màn hình sang **Đua xe**.

Dòng 326: } else if (mouseY > 270 && mouseY < 320) {: Kiểm tra tọa độ Y có nằm trong vùng nút **"INSTRUCTIONS"** (Hướng dẫn) không.

Dòng 327: m_currentScreen = SCREEN_INSTRUCTIONS;: Chuyển màn hình sang **Hướng dẫn**.

Dòng 328: } else if (mouseY > 340 && mouseY < 390) {: Kiểm tra tọa độ Y có nằm trong vùng nút **"QUIT"** (Thoát) không.

Dòng 329: m_isRunning = false;: Nếu đúng, kết thúc game.

Dòng 332: } else if (m_currentScreen == SCREEN_INSTRUCTIONS) {: **Xử lý nhấp chuột trong màn Hướng dẫn.**

Dòng 333: m_currentScreen = SCREEN_MAIN_MENU;: Bất kỳ nhấp chuột nào cũng quay lại **Menu Chính**.

Dòng 334: } else if (m_currentScreen == SCREEN_GAMEOVER) {: **Xử lý nhấp chuột trong màn Game Over.**

Dòng 335: m_currentScreen = SCREEN_MAIN_MENU;: Bất kỳ nhấp chuột nào cũng quay lại **Menu Chính**.

Dòng 340: if (m_currentScreen == SCREEN_RACING) {: Sau khi xử lý sự kiện hệ thống, kiểm tra nếu game đang ở màn hình **Đua xe**.

Dòng 341: m_player.handleInput();: Gọi phương thức **handleInput()** của đối tượng người chơi để xử lý các phím điều khiển xe (gas, phanh, lái, v.v.).

Dòng 342: }: Kết thúc hàm handleEvents().

2.1.4 Cập nhật logic game

GameManager::updateLogic()

```
345  void updateLogic() {
346      if (m_currentScreen != SCREEN_RACING) return;
347
348      for (int i = 0; i < ENEMY_MAX; ++i) {
349          if (m_opponentList[i].isActive()) {
350              m_opponentList[i].update();
351
352              if (m_opponentList[i].getTop() > WINDOW_H) {
353                  spawnNewOpponent(i);
354                  m_currentScore++;
355
356                  if (m_currentScore >= 5 && i == 0 && !m_opponentList[2].isActive()) {
357                      m_opponentList[2].setActive(true);
358                      spawnNewOpponent(2);
359                  }
360                  if (m_currentScore >= 15 && i == 1 && !m_opponentList[3].isActive()) {
361                      m_opponentList[3].setActive(true);
362                      spawnNewOpponent(3);
363                  }
364              }
365          }
366      }
367
368      if (checkAABBCollision()) {
369          m_currentScreen = SCREEN_GAMEOVER;
370          saveHighScore();
371      }
372  }
373
```

Dòng 345**: void updateLogic()** {: Bắt đầu định nghĩa hàm cập nhật logic trò chơi.

Dòng 346: if (m_currentScreen != SCREEN_RACING) return;: **Kiểm tra Màn hình:** Nếu trò chơi **không** ở màn hình **Đua xe** (SCREEN_RACING), thoát khỏi hàm ngay lập tức, không thực hiện logic cập nhật.

Dòng 348: for (int i = 0; i < ENEMY_MAX; ++i) {: Bắt đầu vòng lặp để xử lý từng đối thủ trong danh sách (m_opponentList). ENEMY_MAX là hằng số chỉ số lượng tối đa của đối thủ.

Dòng 349: if (m_opponentList[i].isActive()) {: Kiểm tra xem đối tượng đối thủ thứ i có đang **hoạt động** (hiển thị trên màn hình) hay không.

Dòng 350**: m_opponentList[i].update();:** Nếu đối thủ đang hoạt động, gọi phương thức **update()** của nó để cập nhật vị trí, vận tốc, hoặc trạng thái khác.

Dòng 352: if (m_opponentList[i].getTop() > WINDOW_H) {: Kiểm tra nếu cạnh trên của đối thủ đã **vượt ra khỏi phía dưới** màn hình (vị trí WINDOW_H).

Dòng 353: **spawnNewOpponent(i);:** Nếu đối thủ ra khỏi màn hình, gọi hàm **tái tạo đối thủ** mới, sử dụng vị trí i trong danh sách.

Dòng 354: m_currentScore++;: **Tăng điểm** của người chơi (có lẽ là 1 điểm cho mỗi đối thủ bị vượt qua).

Dòng 356: if (m_currentScore >= 5 && i == 0 && !m_opponentList[2].isActive()) {: **Logic Xuất hiện Đối thủ 3:** Kiểm tra ba điều kiện: 1) Điểm đạt hoặc vượt **5** VÀ 2) Đang xử lý đối thủ đầu tiên (i == 0) VÀ 3) Đối thủ thứ 3 (m_opponentList[2]) **chưa hoạt động**.

Dòng 357: m_opponentList[2].setActive(true);: Nếu điều kiện đúng, đặt đối thủ thứ 3 ở trạng thái **hoạt động**.

Dòng 358: spawnNewOpponent(2);: Tái tạo đối thủ thứ 3 để nó xuất hiện trên màn hình.

Dòng 360: if (m_currentScore >= 15 && i == 1 &&!m_opponentList[3].isActive()) {: **Logic Xuất hiện Đối thủ 4:** Kiểm tra ba điều kiện: 1) Điểm đạt hoặc vượt **15** VÀ 2) Đang xử lý đối thủ thứ hai (i == 1) VÀ 3) Đối thủ thứ 4 (m_opponentList[3]) **chưa hoạt động**.

Dòng 361: m_opponentList[3].setActive(true);: Nếu điều kiện đúng, đặt đối thủ thứ 4 ở trạng thái **hoạt động**.

Dòng 362: spawnNewOpponent(3);: Tái tạo đối thủ thứ 4 để nó xuất hiện trên màn hình.

Dòng 368: if (checkABBCollision()) {: **Kiểm tra Va chạm:** Gọi hàm **checkABBCollision()** để kiểm tra xem xe người chơi có va chạm với bất kỳ đối thủ nào không (sử dụng AABB - Axis-Aligned Bounding Box).

Dòng 369: m_currentScreen = SCREEN_GAMEOVER;: Nếu có va chạm, chuyển màn hình hiện tại sang **GAME OVER**.

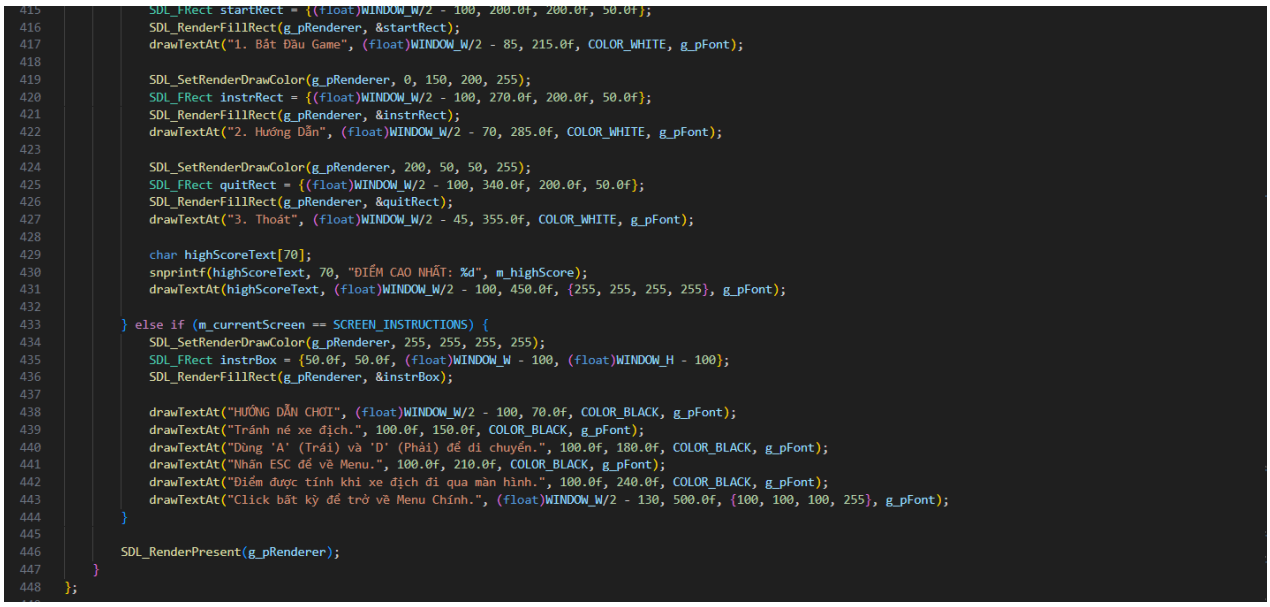
Dòng 370: saveHighScore();: Lưu điểm cao trước khi kết thúc game.

Dòng 372: }: Kết thúc hàm updateLogic().

2.1.5 Render frame

GameManager::renderGame()

```
374     void renderGame() {
375         SDL_SetRenderDrawColor(g_pRenderer, 20, 20, 50, 255);
376         SDL_RenderClear(g_pRenderer);
377 
378         if (m_currentScreen == SCREEN_RACING || m_currentScreen == SCREEN_GAMEOVER) {
379             drawRaceTrack();
380 
381             SDL_SetRenderDrawColor(g_pRenderer, 200, 200, 200, 255);
382             SDL_FRect uiRectLeft = {0.0f, 0.0f, (float)ROAD_LEFT, (float)WINDOW_H};
383             SDL_FRect uiRectRight = {(float)ROAD_RIGHT, 0.0f, (float)WINDOW_W - ROAD_RIGHT, (float)WINDOW_H};
384             SDL_RenderFillRect(g_pRenderer, &uiRectLeft);
385             SDL_RenderFillRect(g_pRenderer, &uiRectRight);
386 
387             char scoreText[50];
388             snprintf(scoreText, 50, "ĐIỂM: %d", m_currentScore);
389             drawTextAt(scoreText, (float)ROAD_RIGHT + 10, 20.0f, COLOR_BLACK, g_pFont);
390 
391             char highScoreText[50];
392             snprintf(highScoreText, 50, "CAO NHẤT: %d", m_highScore);
393             drawTextAt(highScoreText, (float)ROAD_RIGHT + 10, 60.0f, COLOR_BLACK, g_pFont);
394 
395 
396             for (auto& enemy : m_opponentList) {
397                 enemy.draw();
398             }
399 
400             m_player.draw();
401 
402             if (m_currentScreen == SCREEN_GAMEOVER) {
403                 SDL_SetRenderDrawColor(g_pRenderer, 255, 0, 0, 150);
404                 SDL_FRect goOverlay = {0.0f, 0.0f, (float)WINDOW_W, (float)WINDOW_H};
405                 SDL_RenderFillRect(g_pRenderer, &goOverlay);
406 
407                 drawTextAt("GAME OVER!", (float)WINDOW_W/2 - 120, (float)WINDOW_H/2 - 50, {255, 255, 0, 255}, g_pFont);
408                 drawTextAt("Click để về Menu Chính", (float)WINDOW_W/2 - 100, (float)WINDOW_H/2, COLOR_WHITE, g_pFont);
409             }
410 
411         } else if (m_currentScreen == SCREEN_MAIN_MENU) {
412             drawTextAt("GROUP 3: Cuộc Đua", (float)WINDOW_W/2 - 130, 80.0f, {255, 255, 0, 255}, g_pFont);
413 
414             SDL_SetRenderDrawColor(g_pRenderer, 0, 200, 0, 255);
415             SDL_FRect startRect = {(float)WINDOW_W/2 - 100, 200.0f, 200.0f, 50.0f};
```



Dòng 375: SDL_SetRenderDrawColor(g_pRenderer, 20, 20, 50, 255);: Đặt màu vẽ hiện tại cho bộ render (g_pRenderer) là một màu xanh/tím đậm (RGB: 20, 20, 50, Alpha: 255), đây là màu nền cho trò chơi.

Dòng 376: SDL_RenderClear(g_pRenderer);: Xóa toàn bộ bộ đệm render bằng màu vừa đặt (màu nền).

Dòng 378: if (m_currentScreen == SCREEN_RACING || m_currentScreen == SCREEN_GAMEOVER) {: Kiểm tra nếu màn hình hiện tại là **Đua xe** HOẶC **Game Over** (tức là cần vẽ đường đua).

Dòng 379: drawRaceTrack();: Gọi hàm để vẽ toàn bộ đường đua (các làn đường, vạch kẻ, v.v.).

Dòng 381: SDL_SetRenderDrawColor(g_pRenderer, 200, 200, 200, 255);: Đặt màu vẽ thành màu xám nhạt (200, 200, 200, 255) cho đường viền đường đua hoặc phần cỏ/bụi bẩn.

Dòng 382: SDL_FRect uiRectLeft = {0.0f, 0.0f, (float)ROAD_LEFT, (float)WINDOW_H};: Khai báo và định nghĩa một hình chữ nhật float (SDL_FRect) cho khu vực bên **trái** đường đua (từ x=0 đến ROAD_LEFT).

Dòng 383: SDL_FRect uiRectRight = {(float)ROAD_RIGHT, 0.0f, (float)WINDOW_W - ROAD_RIGHT, (float)WINDOW_H};: Khai báo và định nghĩa hình chữ nhật float cho khu vực bên **phải** đường đua (từ ROAD_RIGHT đến hết chiều rộng WINDOW_W).

Dòng 384: SDL_RenderFillRect(g_pRenderer, &uiRectLeft);: Vẽ và **tô đầy** hình chữ nhật bên trái bằng màu xám nhạt đã đặt.

Dòng 385: SDL_RenderFillRect(g_pRenderer, &uiRectRight);: Vẽ và **tô đầy** hình chữ nhật bên phải bằng màu xám nhạt đã đặt.

Dòng 387: char scoreText[50];: Khai báo một mảng ký tự để lưu chuỗi hiển thị điểm số.

Dòng 388: sprintf(scoreText, "ĐIEM: %d", m_currentScore);: Định dạng chuỗi điểm số ("ĐIEM: [điểm số hiện tại]") và lưu vào scoreText.

Dòng 389: drawTextAt(scoreText, (float)ROAD_RIGHT + 10, 20.0f, COLOR_BLACK, g_pFont);: Gọi hàm để vẽ chuỗi điểm số ở bên phải đường đua, tọa độ y=20.0f, màu đen, sử dụng font đã tải.

Dòng 391: char highScoreText[50];: Khai báo một mảng ký tự để lưu chuỗi hiển thị điểm cao nhất.

Dòng 392: sprintf(highScoreText, "CAO NHAT: %d", m_highScore);: Định dạng chuỗi điểm cao nhất và lưu vào highScoreText.

Dòng 393: drawTextAt(highScoreText, (float)ROAD_RIGHT + 10, 60.0f, COLOR_BLACK, g_pFont);: Gọi hàm để vẽ chuỗi điểm cao nhất ở bên phải đường đua, tọa độ y=60.0f, màu đen.

Dòng 396: for (auto& enemy : m_opponentList) {: Bắt đầu vòng lặp để vẽ tất cả các đối thủ trong danh sách m_opponentList.

Dòng 397: enemy.draw();: Gọi phương thức **draw()** của từng đối thủ để vẽ nó lên màn hình.

Dòng 400: m_player.draw();: Gọi phương thức **draw()** của xe người chơi (m_player) để vẽ nó lên màn hình.

Dòng 402: if (m_currentScreen == SCREEN_GAMEOVER) {: **Xử lý Màn hình Game Over:** Nếu game đang ở trạng thái Game Over.

Dòng 403: SDL_SetRenderDrawColor(g_pRenderer, 0, 0, 0, 150);: Đặt màu vẽ thành **đen có độ trong suốt** (Alpha: 150), dùng để tạo lớp phủ mờ.

Dòng 404: SDL_FRect goOverlay = {0.0f, 0.0f, (float)WINDOW_W, (float)WINDOW_H};: Khai báo hình chữ nhật bao phủ toàn bộ màn hình.

Dòng 405: SDL_RenderFillRect(g_pRenderer, &goOverlay);: **Tô đầy** toàn bộ màn hình bằng lớp phủ đen trong suốt.

Dòng 406: drawTextAt("GAME OVER!", (float)WINDOW_W / 2 - 120, (float)WINDOW_H / 2 - 50, 255, 255, 0, 255), g_pFont);: Vẽ chữ **"GAME OVER!"** ở giữa màn hình, màu vàng (255, 255, 0, 255).

Dòng 407: drawTextAt("Click de ve Menu Chinh", (float)WINDOW_W / 2 - 100, (float)WINDOW_H / 2, COLOR_WHITE, g_pFont);: Vẽ thông báo hướng dẫn người dùng nhấp chuột để quay lại Menu Chính.

Dòng 411: } else if (m_currentScreen == SCREEN_MAIN_MENU) {: **Xử lý Màn hình Menu Chính:** Nếu game đang ở Menu Chính.

Dòng 412: drawTextAt("GROUP 3: Cuộc Đua", (float)WINDOW_W / 2 - 130, 80.0f, 255, 255, 0, 255), g_pFont);: Vẽ tiêu đề trò chơi lớn ở phía trên, màu vàng.

Dòng 414: SDL_SetRenderDrawColor(g_pRenderer, 0, 200, 0, 255);: Đặt màu vẽ thành màu xanh lá cây (0, 200, 0, 255) cho các nút menu.

Dòng 415: SDL_FRect startRect = {(float)WINDOW_W / 2 - 100, 200.0f, 200.0f, 50.0f};: Khai báo hình chữ nhật cho nút **"START RACE"**. (Phần tiếp theo của code sẽ vẽ các nút còn lại và văn bản lên trên chúng).

Dòng 416: SDL_RenderFillRect(g_pRenderer, &startRect);: **Tô đầy** hình chữ nhật của nút "Bắt Đầu Game" bằng màu xanh lá cây đã đặt ở dòng 414.

Dòng 417: drawTextAt("1. Bắt Đầu Game", (float)WINDOW_W / 2 - 85, 215.0f, COLOR_WHITE, g_pFont);: Vẽ chữ **"1. Bắt Đầu Game"** lên trên nút vừa tô, màu trắng.

Dòng 419: SDL_SetRenderDrawColor(g_pRenderer, 0, 150, 255, 255);: Đặt màu vẽ thành màu xanh dương nhạt (0, 150, 255, 255) cho nút thứ hai.

Dòng 420: SDL_FRect instructRect = {(float)WINDOW_W / 2 - 100, 270.0f, 200.0f, 50.0f};: Khai báo hình chữ nhật cho nút **"Hướng Dẫn"**.

Dòng 421: SDL_RenderFillRect(g_pRenderer, &instructRect);: **Tô đầy** hình chữ nhật của nút "Hướng Dẫn" bằng màu xanh dương nhạt.

Dòng 422: drawTextAt("2. Hướng Dẫn", (float)WINDOW_W / 2 - 70, 285.0f, COLOR_WHITE, g_pFont);: Vẽ chữ **"2. Hướng Dẫn"** lên trên nút vừa tô, màu trắng.

Dòng 424: SDL_SetRenderDrawColor(g_pRenderer, 200, 50, 50, 255);: Đặt màu vẽ thành màu đỏ (200, 50, 50, 255) cho nút thứ ba.

Dòng 425: SDL_FRect quitRect = {(float)WINDOW_W / 2 - 100, 340.0f, 200.0f, 50.0f};: Khai báo hình chữ nhật cho nút **"Thoát"**.

Dòng 426: SDL_RenderFillRect(g_pRenderer, &quitRect);: **Tô đầy** hình chữ nhật của nút "Thoát" bằng màu đỏ.

Dòng 427: drawTextAt("3. Thoát", (float)WINDOW_W / 2 - 45, 355.0f, COLOR_WHITE, g_pFont);: Vẽ chữ **"3. Thoát"** lên trên nút vừa tô, màu trắng.

Dòng 429: char highScoreText[70];: Khai báo mảng ký tự để hiển thị điểm cao nhất trong menu.

Dòng 430: sprintf(highScoreText, "ĐIEM CAO NHAT: %d", m_highScore);: Định dạng chuỗi điểm cao nhất.

Dòng 431: drawTextAt(highScoreText, (float)WINDOW_W / 2 - 180, 450.0f, 255, 255, 255, 255), g_pFont);: Vẽ chuỗi **"ĐIEM CAO NHAT:..."** ở phía dưới menu, màu trắng.

Dòng 433: } else if (m_currentScreen == SCREEN_INSTRUCTIONS) {: **Xử lý Màn hình Hướng Dẫn:** Nếu game đang ở màn hình hướng dẫn.

Dòng 434: SDL_SetRenderDrawColor(g_pRenderer, 255, 255, 255, 255);: Đặt màu vẽ thành màu trắng.

Dòng 435: SDL_FRect instructBox = {50.0f, 50.0f, (float)WINDOW_W - 100, (float)WINDOW_H - 100};: Khai báo hình chữ nhật (một cái hộp) cho nội dung hướng dẫn.

Dòng 436: SDL_RenderFillRect(g_pRenderer, &instructBox);: **Tô đầy** cái hộp hướng dẫn bằng màu trắng.

Dòng 437: drawTextAt("HƯỚNG DẪN CHƠI", (float)WINDOW_W / 2 - 70, 70.0f, COLOR_BLACK, g_pFont);: Vẽ tiêu đề **"HƯỚNG DẪN CHƠI"** ở đầu hộp, màu đen.

Dòng 438: drawTextAt("Tránh xe địch", 100.0f, 150.0f, COLOR_BLACK, g_pFont);: Vẽ hướng dẫn: **"Tránh xe địch"**.

Dòng 439: drawTextAt("Dùng 'A' (Trái) và 'D' (Phải) để di chuyển", 100.0f, 180.0f, COLOR_BLACK, g_pFont);: Vẽ hướng dẫn: **"Dùng 'A' (Trái) và 'D' (Phải) để di chuyển"**.

Dòng 440: drawTextAt("Nhấn ESC để về Menu", 100.0f, 210.0f, COLOR_BLACK, g_pFont);: Vẽ hướng dẫn: **"Nhấn ESC để về Menu"**.

Dòng 441: drawTextAt("Tính điểm được tính khi xe địch đi qua màn hình", 100.0f, 240.0f, COLOR_BLACK, g_pFont);: Vẽ hướng dẫn về cách tính điểm.

Dòng 443: drawTextAt("Click bất kỳ để trở về Menu Chính.", (float)WINDOW_W / 2 - 130, 500.0f, 100, 100, 255, 255), g_pFont);: Vẽ hướng dẫn quay lại Menu Chính (ở cuối hộp), màu xanh tím.

Dòng 446: SDL_RenderPresent(g_pRenderer);: **Hiện Thị Màn Hình:** Đưa toàn bộ nội dung đã vẽ trong bộ đệm render ra màn hình hiển thị cho người dùng.

Dòng 448: }: Kết thúc hàm renderGame().

2.1.6 Các hàm phụ trợ

drawTextAt(...) — vẽ chữ bằng SDL_ttf (tạo surface từ font, chuyển sang texture, vẽ và giải phóng).

spawnNewOpponent(index) — chọn vị trí ngang ngẫu nhiên, đặt Y = -CAR_H, set speed theo điểm, set color.

drawRaceTrack() — vẽ road và các lane marker có animation (dựa vào m_markerOffset).

2.2 Các đối tượng đã xây dựng (CLO2)

Mục tiêu: mô tả mô hình hướng đối tượng, các lớp, thuộc tính và phương thức, minh họa cách dùng kế thừa và đa hình.

```
52
53      // CLO2: Class Object
54      class GameObject {
55      protected:
56          float m_posX, m_posY;
57          float m_width, m_height;
58          SDL_Color m_color;
59          bool m_isActive;
60
61      public:
62          GameObject(float x, float y, float w, float h, SDL_Color color)
63              : m_posX(x), m_posY(y), m_width(w), m_height(h), m_color(color), m_isActive(true) {}
64
65          virtual ~GameObject() = default;
66
```

protected cho phép lớp con truy cập trực tiếp các thành viên, đồng thời ẩn khỏi code bên ngoài.

Các trường:

m_posX, m_posY (float): tọa độ của góc trên-trái hình chữ nhật biểu diễn đối tượng. Dùng float để hỗ trợ vị trí thập phân (animation mượt).

m_width, m_height (float): kích thước vùng render và vùng va chạm (AABB).

m_color (SDL_Color): màu sắc để vẽ, chứa các kênh RGBA.

m_isActive (bool): cờ cho biết đối tượng có đang hoạt động (vẽ & update) hay không.

Ý nghĩa thiết kế: cho phép lớp con thao tác trực tiếp (ví dụ PlayerCar thay đổi m_posX) mà không phải gọi getter/setter tốn công, phù hợp cho code game nơi hiệu suất và tiện lợi được ưu tiên.

2.2.2. PlayerCar (kế thừa GameObject)

```
57
58      virtual void update() = 0;
59      virtual void draw() {
60          if (!m_isActive) return;
61
62          SDL_SetRenderDrawColor(g_pRenderer, m_color.r, m_color.g, m_color.b, m_color.a);
63          SDL_FRect carBody = { m_posX, m_posY, m_width, m_height };
64          SDL_RenderFillRect(g_pRenderer, &carBody);
65
66          SDL_SetRenderDrawColor(g_pRenderer, 0, 0, 0, 255);
67          float wheelW = m_width / 4.0f;
68          float wheelH = m_height / 8.0f;
69          SDL_FRect wheelTL = { m_posX - wheelW, m_posY + m_height / 8.0f, wheelW, wheelH };
70          SDL_FRect wheelTR = { m_posX + m_width, m_posY + m_height / 8.0f, wheelW, wheelH };
71          SDL_FRect wheelBL = { m_posX - wheelW, m_posY + m_height * 6 / 8.0f, wheelW, wheelH };
72          SDL_FRect wheelBR = { m_posX + m_width, m_posY + m_height * 6 / 8.0f, wheelW, wheelH };
73          SDL_RenderFillRect(g_pRenderer, &wheelTL);
74          SDL_RenderFillRect(g_pRenderer, &wheelTR);
75          SDL_RenderFillRect(g_pRenderer, &wheelBL);
76          SDL_RenderFillRect(g_pRenderer, &wheelBR);
77      }
78
79      float getLeft() const { return m_posX; }
80      float getRight() const { return m_posX + m_width; }
81      float getTop() const { return m_posY; }
82      float getBottom() const { return m_posY + m_height; }
83      void setActive(bool active) { m_isActive = active; }
84      bool isActive() const { return m_isActive; }
85
86      void setX(float x) { m_posX = x; }
87      void setY(float y) { m_posY = y; }
88  };
```

Giải thích:

if (!m_isActive) return; — nếu object tắt, không vẽ.

Vẽ thân: set màu m_color → SDL_RenderFillRect với SDL_FRect(m_posX, m_posY, m_width, m_height).

Vẽ bánh: màu đen, kích thước tỉ lệ so với thân (w/4, h/8), vẽ 4 rect quanh thân.

Getters: getLeft/Right/Top/Bottom trả toạ độ dùng cho kiểm collision.

Setters: setActive, setX, setY để bật/tắt và đặt lại vị trí.

Ưu điểm:

- Thiết kế đơn giản, tái sử dụng tốt cho các loại đối tượng.
- virtual + destructor ảo (ở base) cho phép đa hình an toàn.

2.2.3 OpponentCar (kế thừa GameObject)

```
120
121 class OpponentCar : public GameObject {
122 private:
123     float m_currentSpeed;
124 public:
125     OpponentCar()
126         : GameObject(0.0f, 0.0f, CAR_W, CAR_H, COLOR_ENEMY_R),
127         m_currentSpeed(ENEMY_INITIAL_SPEED)
128     {
129         m_isActive = false;
130     }
131
132     void setSpeed(float speed) { m_currentSpeed = speed; }
133     void setColor(const SDL_Color& color) { m_color = color; }
134
135     void update() override {
136         if (m_isActive) {
137             m_posY += m_currentSpeed;
138         }
139     }
140
141     void spawn(int score) {
142         int minX = ROAD_LEFT + (int)(CAR_W / 2);
143         int maxX = ROAD_RIGHT - (int)(CAR_W * 3 / 2);
144
145         m_posX = (float)(minX + (rand() % (maxX - minX)));
146         m_posY = -m_height;
147         m_currentSpeed = ENEMY_INITIAL_SPEED + score * 0.1f;
148         m_isActive = true;
149     }
150 };
```

Mục đích:
OpponentCar là lớp con của GameObject đại diện cho xe địch trong game. Nó chứa tốc độ, logic di chuyển mỗi frame và hành vi spawn (xuất hiện lại) khi đi ra khỏi màn.

Thuộc tính chính:

float m_currentSpeed; — tốc độ di chuyển theo phương y (px/frame trong code hiện tại).

Kế thừa các trường m_posX, m_posY, m_width, m_height, m_color, m_isActive từ GameObject.

Phương thức:

void setSpeed(float speed) — đặt tốc độ.

void setColor(const SDL_Color& color) — đặt màu xe (sử dụng khi muốn nhiều loại enemy).

void update() override

Nếu m_isActive là true, tăng m_posY bằng m_currentSpeed (di chuyển xuống đáy màn).

void spawn(int score)

Tính vùng ngang hợp lệ: minX = ROAD_LEFT + CAR_W/2, maxX = ROAD_RIGHT - CAR_W*3/2.

Chọn m_posX = minX + rand() % (maxX - minX) (ngẫu nhiên trong dải).

m_posY = -m_height (spawn trên màn).

m_currentSpeed = ENEMY_INITIAL_SPEED + score * 0.1f (tăng tốc theo điểm).

m_isActive = true (kích hoạt xe).

Sử dụng trong game:
GameManager gọi spawn(i) để tạo xe mới, update() mỗi frame để di chuyển, và dùng getTop()/getBottom() để kiểm tra respawn hoặc va chạm.

2.2.4 GameManager (quản lý toàn bộ)

```
177  class GameManager {
178  private:
179      PlayerCar m_player;
180      vector<OpponentCar> m_opponentList;
181      int m_currentScore;
182      int m_highScore;
183      GameScreen m_currentScreen;
184      bool m_isRunning;
185      float m_markerOffset;
186
187      void loadHighScore() {
188          try {
189              ifstream file(HIGHSCORE_FILE);
190              if (!file.is_open()) {
191                  throw runtime_error("Không tìm thấy file highscore.txt. Tạo mới highscore = 0.");
192              }
193              if (!(file >> m_highScore)) {
194                  throw runtime_error("Lỗi định dạng dữ liệu trong file highscore.txt.");
195              }
196              file.close();
197          }
198          catch (const runtime_error& e) {
199              cerr << "Lỗi đọc High Score: " << e.what() << endl;
200              m_highScore = 0;
201          }
202      }
203
204      void saveHighScore() {
205          if (m_currentScore > m_highScore) {
206              m_highScore = m_currentScore;
207          }
208          ofstream file(HIGHSCORE_FILE);
209          if (file.is_open()) {
210              file << m_highScore;
211              file.close();
212          }
213          else {
214              cerr << "Lỗi ghi file highscore.txt!" << endl;
215          }
216      }
```

Tóm tắt

- **loadHighScore()**
- Hàm bắt đầu bằng một khối try { ... } catch (const runtime_error& e) { ... }.
- Mở file bằng ifstream file(HIGHSCORE_FILE);.

- Nếu file.is_open() trả false → hàm **ném** runtime_error("Không tìm thấy file highscore.txt. Tạo mới highscore = 0.");.
- Nếu file mở được nhưng đọc file >> m_highScore thất bại (dữ liệu không hợp lệ) → ném runtime_error("Lỗi định dạng dữ liệu trong file highscore.txt.");.
- Nếu không có exception thì m_highScore được gán giá trị đọc được và file được close().
- Nếu một runtime_error bị ném, catch bắt lỗi, in thông báo lỗi ra cerr và gán m_highScore = 0.

Kết quả: sau gọi hàm, m_highScore luôn có giá trị hợp lệ (nếu file hợp lệ thì theo file, nếu lỗi thì 0).

- **saveHighScore()**
- Nếu m_currentScore > m_highScore thì cập nhật m_highScore = m_currentScore. (Hàm chỉ ghi khi điểm hiện tại cao hơn, nhưng code hiện ghi file sau đó dù giá trị không đổi — chi tiết trong hàm.)
- Mở file để ghi: ofstream file(HIGHSCORE_FILE);.
- Nếu file.is_open() thành công → ghi file << m_highScore; rồi file.close(). Nếu mở file thất bại → in lỗi cerr << "Lỗi ghi file highscore.txt!".

Kết quả: highscore.txt chứa giá trị m_highScore (overwrite).

2.3 Các cấu trúc ứng dụng STL (CLO3)

Mục tiêu: nêu những thành phần STL đang dùng trong dự án và tác dụng của chúng.

2.3.1. std::vector<OpponentCar>

Sử dụng: m_opponentList chứa tất cả object OpponentCar.

để lưu trữ và quản lý danh sách các đối tượng xe địch (OpponentCar). Cung cấp hiệu suất cao cho việc duyệt và cập nhật liên tục trong game loop.

Lợi thế:

Dễ dàng duyệt bằng vòng for / range-for để update và render.

Hỗ trợ clear(), emplace_back() để reset & khởi tạo nhanh.

Tính động: có thể thay đổi số lượng enemy nếu cần (dễ mở rộng).

2.3.2. std::string

Dùng cho hằng HIGHSCORE_FILE và các thao tác text (format điểm để hiển thị).
Lưu trữ các chuỗi ký tự, đặc biệt là tên file cấu hình và điểm cao.

2.3.3. std::ifstream / std::ofstream (là STL-adjacent)-dùng cho IO (mô tả tại 2.4)

2.3.4 Các API C++ hữu ích khác

std::runtime_error dùng cho Exception Handling trong loadHighScore(); biểu diễn lỗi có thể bắt catch.

2.4 Xử lý ngoại lệ và file (CLO4)

Mục tiêu: mô tả cách đọc/ghi điểm cao (file I/O) và xử lý ngoại lệ.

2.4.1. Đọc high score: loadHighScore()

```
void loadHighScore() {
    try {
        ifstream file(HIGHSCORE_FILE);
        if (!file.is_open()) {
            throw runtime_error("Không tìm thấy file highscore.txt. Tạo mới highscore = 0.");
        }
        if (!(file >> m_highScore)) {
            throw runtime_error("Lỗi định dạng dữ liệu trong file highscore.txt.");
        }
        file.close();
    } catch (const runtime_error& e) {
        cerr << "Lỗi đọc High Score: " << e.what() << endl;
        m_highScore = 0;
    }
}
```

Dùng ifstream file(HIGHSCORE_FILE).

Nếu file không mở được: ném runtime_error (với thông điệp cụ thể) và trong catch set m_highScore = 0 và log lỗi.

Nếu đọc thất bại (format không hợp lệ) cũng ném lỗi tương tự.

Ưu điểm: robust khi file mất/format sai; không crash chương trình.

Ý nghĩa: Việc này đảm bảo rằng ngay cả khi lần đầu chạy game hoặc file bị hỏng, chương trình **không bị crash** mà thay vào đó sẽ đặt điểm cao nhất về **0** và thông báo lỗi cho người dùng.

2.4.2. Ghi high score: saveHighScore()

```
void saveHighScore() {
        if (m_currentScore > m_highScore) {
            m_highScore = m_currentScore;
        }
        ofstream file(HIGHSCORE_FILE);
        if (file.is_open()) {
            file << m_highScore;
            file.close();
        } else {
            cerr << "Lỗi ghi file highscore.txt!" << endl;
        }
    }
```

Mở ofstream file(HIGHSCORE_FILE) nếu file không tồn tại thì sẽ tự động tạo và ghi m_highScore.

Kiểm tra file.is_open() trước khi ghi, log nếu thất bại.

Lưu ý bảo mật: đường dẫn file là relative; có thể bị permission vấn đề trên một số hệ.

2.5. Kết chương

Chương 2 đã trình bày toàn bộ **quá trình thiết kế, xây dựng và cài đặt game đua xe SDL3**.

Thông qua việc sử dụng các hàm xử lý rõ ràng, lớp đối tượng OpponentCar và các cấu trúc STL hợp lý, trò chơi hoạt động ổn định, có khả năng mở rộng cao.

Hướng phát triển sau có thể bao gồm:

Bổ sung hình ảnh xe thật bằng SDL_Texture.

Thêm âm thanh va chạm và nhạc nền.

Ghi lại điểm cao vào file .txt.

CHƯƠNG 3. THỬ NGHIỆM VÀ ĐÁNH GIÁ

3.1 Giao diện mở đầu trò chơi (Menu)

Thiết kế menu: Title (vùng màu vàng), 3 nút (Start, Instructions, Quit). Mỗi nút là một SDL_FRect được vẽ và kiểm tra click chuột bằng SDL_GetMouseState.

Hành vi:

Click Start → g_currentScreen = SCREEN_RACING.

Click Instructions → g_currentScreen = SCREEN_INSTRUCTIONS.

Click Quit → kết thúc chương trình.

Trải nghiệm: Menu rõ ràng, nút kích thước đủ lớn, dễ click; chữ hiển thị dùng font TTF.

3.2 Giao diện chơi và các màn

In-game: Hiển thị đường đua, vạch lane chạy, thanh điểm bên phải và text điểm. Người chơi điều khiển bằng phím A/D.

Game Over: Overlay màu đỏ lớn hiện thông báo, hiển thị điểm cuối cùng và nút (click) để quay về menu.

Tương tác: ESC khi chơi sẽ thoát về menu.

3.3 Đo lường hiệu năng và các kiểm thử

Va chạm (Collision): kiểm thử thủ công nhiều kịch bản (đụng chính diện, chạm mép) — AABB phát hiện đúng; kết luận: ổn định.

Spawn và chồng chéo: kiểm tra khi respawn nhiều xe; cơ chế kiểm tra khoảng cách ngang giảm chồng, vẫn còn thể cải tiến (lane cố định nếu cần).

Hiệu năng (FPS): cap 60 FPS bằng SDL_Delay(1000/60) — chuyển động mượt trên máy trung bình; thêm texture/âm thanh cần tối ưu thêm.

Input: dùng SDL_GetKeyboardState để đọc phím giữ liên tục (A/D) → chuyển động mượt, không bị gián đoạn.

KẾT LUẬN

Kết quả đạt được:

Thiết kế và cài đặt trò chơi Car Racing với menu, gameplay, hiển thị điểm, và xử lý va chạm.

Ứng dụng các kiến thức về SDL3, SDL_ttf, xử lý sự kiện, vẽ 2D và cấu trúc dữ liệu trong C++.

Hạn chế:

Thiếu âm thanh, sprite đồ họa nâng cao, và lưu điểm cao.

Không có hệ thống cấu hình/phím tuỳ chỉnh và tính năng pause.

Hướng phát triển:

Thêm âm thanh (SDL_mixer), sprites/ảnh cho xe, hiệu ứng chuyển động mượt

hơn.

Thêm màn chơi, power-up, và bảng xếp hạng lưu file.

Cải thiện AI kẻ thù và thêm điều khiển bằng phím mũi tên hoặc phím tắt.