Lập trình Game với SDL 2

Mục lục

Mục tiêu

Giới thiệu về thư viện đồ họa SDL 2, cách thức cài đặt thư viện, các nhóm hàm cơ bản để có thể triển khai một ứng dụng đồ họa 2D đơn giản. Xây dựng bộ đếm *frame-per-second*. Tìm hiểu nguyên lý và xây dựng các trò chơi đơn giản như Pong (bóng bàn) và Snake (rắn săn mồi).

Yêu cầu

Các kiến thức cơ sở về ngôn ngữ lập trình C/C++. Nắm vững kỹ thuật lập trình cùng với một số cấu trúc dữ liệu đơn giản như mảng, danh sách, hàng đợi...

Thư viện lập trình đồ họa SDL 2

Giới thiệu

SDL 2 (Simple DirectMedia Layer version 2) là một thư viện lập trình có khả năng trừu tượng hóa các phần cứng đồ họa, âm thanh, thiết bị I/O. Sử dụng SDL, các lập trình viên có thể xây dựng các ứng dụng giải trí, trò chơi điện tử, ứng dụng truyền thông đa phương tiện trên nhiều hệ điều hành khác nhau như Linux, macOS, Windows...

Thư viện lập trình SDL 2 được chia thành nhiều hệ thống nhỏ hơn (sub-system) như Video, Audio, CD-ROM, Timer... Ngoài các hệ thống con cơ bản này, SDL 2 còn cung cấp thêm các thư viện chính thức riêng biệt khác như:

- `SDL_image`: cho phép đọc các định dạng ảnh phổ biến như JPEG, PNG, BMP...
- `SDL_mixer`: cung cấp các hàm audio cho phép thực hiện việc hòa âm (mixing), đọc các tập tin audio như OGG, MP3...
- `SDL_net`: hỗ trợ lập trình mạng.
- `SDL_ttf`: hỗ trợ sử dụng các font chữ TrueType.
- `SDL_rtf`: hỗ trợ hiển thị tài liệu dạng Rich Text Format (RTF).

SDL 2 được xây dựng bằng ngôn ngữ lập trình C, vì vậy C cũng chính là ngôn ngữ được hỗ trợ chính thức bởi thư viện lập trình này, ngoài ra ta cũng có thể sử dụng SDL 2 với C++.

Cài đặt SDL 2

SDL 2 hiện tại hỗ trợ các trình biên dịch như Visual Studio và MinGW. Trong nội dung bài hướng dẫn này, ta chỉ quan tâm đến việc cài đặt SDL 2.0 vào Visual Studio, cụ thể là phiên bản 2015. Đối với các phiên bản cũ hoặc mới hơn, quá trình cài đặt thư viện cũng tương tự, chỉ có một số điều chỉnh nhỏ do sự khác biệt về bố trí thư mục cài đặt của các phiên bản Visual Studio là không giống nhau.

Việc cài đặt SDL 2 vào hệ thống sẽ giúp cho việc xây dựng, biên dịch và liên kết các dự án phần mềm có sử dụng thư viện này được thuận tiện và dễ dàng.

Trước tiên, ta cần tải về bộ thư viện lập trình SDL 2 trực tiếp từ trang chủ tại địa chỉ: `https://www.libsdl.org/download-2.0.php`. Tại đây, ta có thể tải về mã nguồn của thư viện, các thư viện liên kết động để sử dụng với các ứng dụng SDL 2 và bộ thư viện hỗ trợ phát triển ứng dụng (*development library*). Do cần sử dụng SDL 2 để xây dựng các ứng dụng trên môi trường Visual Studio, ta tải về tập tin chứa thư viện phát triển của SDL phiên bản 2.0.8: `SDL2-devel-2.0.8-VC.zip` trên nền Windows.

Sau khi tải về tập tin trên, giải nén nội dung tập tin này vào một thư mục và thực hiện các bước cài đặt như sau:

- Giả sử Visual Studio 2015 được cài đặt trên ổ đĩa `C:` của hệ điều hành tại thư mục `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC`. Khi đó thư mục chứa các tập tin header của Visual C++ sẽ được đặt trong thư mục `include` và thư mục chứa các thư viện liên kết sẽ được đặt trong thư mục `lib`, (Hình 1). Thư mục `Windows` cũng được cài đặt trên ổ đĩa `C:` của hệ thống. Các thư mục cần thao tác trong quá trình cài đặt thư viện SDL 2 được đánh dấu `(x)`.
- Tạo thư mục `SDL2` bên trong thư mục `include` của Visual C++, sau đó copy toàn bộ các tập tin header (`.h`) trong thư mục `include` của thư viện SDL 2 vừa giải nén vào thư mục `include/SDL2` vừa tạo. Thao tác này cho phép ta sử dụng các tập tin header của SDL2 thông qua chỉ thị `#include` ở đầu tập tin mã nguồn của mọi project phần mềm mà không cần phải sao chép chúng vào từng thư mục riêng biệt của project đó.
- Tương tự, copy tất cả các tập tin `.lib` từ thư mục `lib/x86` của thư viện SDL 2 đã giải nén vào thư mục `lib` của Visual C++. Copy tất cả các tập tin `.lib` từ thư mục

`lib/x64` vào thư mục `lib/amd64` của Visual C++. Để liên kết các thư viện SDL 2 vào dự án phần mềm, ta sử dụng chỉ thị `#pragma` trực tiếp trong tập tin mã nguồn của dự án.

- Copy tập tin `SDL2.dll` từ thư mục `lib/x86` vào thư mục `Windows/SysWOW64` và tập tin `SDL2.dll` từ thư mục `lib/x64` vào thư mục `Windows/System32`.

```
C:
├── Program Files (x86)
│   └── Microsoft Visual Studio 14.0      // thư mục cài đặt VS-2015.
│       └── VC                            // thư mục Visual C++
│           ├── include                   // chứa các tập tin .h
│           │   └── SDL2           (x)    // ==> ta tạo thư mục này
│           └── lib                (x)    // chứa các thư viện .lib 32-bit
│               └── amd64          (x)    // chứa các thư viện .lib 64-bit
└── Windows                               // thư mục Windows
    ├── System32                   (x)    // chứa các tập tin .dll 64-bit
    └── SysWOW64                   (x)    // chứa các tập tin .dll 32-bit
```

Hình 1. Cấu trúc thư mục Visual Studio 2015 và Windows trên ổ đĩa C:

Sử dụng SDL 2

Cách thức vận dụng các hàm API của thư viện SDL 2 được giới thiệu chi tiết tại trang wiki của thư viện này tại địa chỉ: `https://wiki.libsdl.org`. Bài hướng dẫn này chỉ trình bày sơ lược các hàm API sẽ được sử dụng trong các bài tập sẽ được giới thiệu mà thôi.

a. Khởi tạo môi trường đồ họa

Để khởi tạo thư viện, xây dựng, hủy bỏ các cửa sổ dựng hình và chấm dứt phiên làm việc với SDL 2, ta sử dụng các hàm sau đây:

- `int SDL_Init(Uint32 flags)`

  Khởi tạo thư viện SDL 2. Hàm này thực chất sẽ thực hiện các công việc khởi tạo cho từng thư viện con ứng với giá trị nhận được từ tham số `flags`. Giá trị thường hay được sử dụng của tham số `flags` là `SDL_INIT_EVERYTHING`. Ta có thể kết hợp việc khởi tạo nhiều thư viện con thông qua toán tử `|`. Ví dụ, nếu ta chỉ cần khởi tạo thư viện con xử lý đồ họa và âm thanh, giá trị `flags` sẽ là `SDL_INIT_VIDEO | SDL_INIT_AUDIO`. Hàm này trả về trị 0 nếu việc khởi tạo được thực hiện thành công.

- `void SDL_Quit(void)`

  Sử dụng hàm này để dọn dẹp và thu hồi tài nguyên đã cấp phát cho các thư viện con và chấm dứt phiên làm việc với SDL 2. Hàm này không nhận tham số và cũng không trả ra bất kỳ giá trị nào.

- ```
  SDL_Window * SDL_CreateWindow( const char * title,
                                 int x, int y, int w, int h,
                                 Uint32 flags)
  ```

  Hàm này thực hiện việc tạo ra một cửa sổ SDL 2 với các tham số như sau:

- `title`: Tiêu đề của cửa sổ SDL.
- `x, y`: tọa độ của cửa sổ. Có thể là một giá trị cụ thể hoặc một trong các hằng số `SDL_WINDOWPOS_CENTERED` hoặc `SDL_WINDOWPOS_UNDEFINED`.
  ```

- `w, h`: Kích thước chiều rộng và chiều cao của cửa sổ, tính theo pixel.
- `flags`: Cờ quy định trạng thái cửa sổ sẽ được tạo ra, một số giá trị có thể sử dụng bao gồm, ta có thể kết hợp các giá trị này thông qua toán tử `|`:
    - `SDL_WINDOW_SHOWN`: Cửa sổ được tạo ra sẽ xuất hiện trên màn hình, giá trị này là mặc định.
    - `SDL_WINDOW_FULLSCREEN`: Cửa sổ được tạo ra sẽ có độ phân giải toàn màn hình.
    - `SDL_WINDOW_FULLSCREEN_DESKTOP`: Cửa sổ được tạo ra sẽ chiếm toàn bộ màn hình desktop, không có thanh tiêu đề, không có viền cửa sổ và các nút chức năng.
    - `SDL_WINDOW_OPENGL`: Cửa sổ được tạo ra sẽ hỗ trợ dựng hình với OpenGL.

Hàm này khi thực hiện thành công sẽ trả về một con trỏ đến cửa sổ đã được tạo ra. Ngược lại, một giá trị `NULL` sẽ được trả về.

- `void SDL_DestroyWindow(SDL_Window * wnd)`

  Ta gọi hàm này để hủy bỏ cửa sổ SDL đã được tạo bởi hàm `SDL_CreateWindow`, trỏ đến bởi tham số `wnd`. Cửa sổ sẽ được hủy bỏ và thu hồi các tài nguyên đã cấp phát.

Ví dụ:

```
#include "SDL.h"
#include <stdio.h>
int main(int argc, char* argv[])
{
    SDL_Window *window;            // Khai báo con trỏ đến cửa sổ SDL
    SDL_Init(SDL_INIT_VIDEO);      // Khởi tạo thư viện con đồ họa

    // Tạo một cửa sổ với các tham số như sau
    window = SDL_CreateWindow(
        "Cửa sổ SDL 2",            // tiêu đề
        SDL_WINDOWPOS_UNDEFINED,   // tọa độ x
        SDL_WINDOWPOS_UNDEFINED,   // tọa độ y
        640, 480,                  // chiều rộng, cao (pixels)
        SDL_WINDOW_SHOWN);

    // Kiểm tra việc tạo cửa sổ có thành công hay không
    if (window == NULL) {
        printf("Không thể khởi tạo cửa sổ: %s\n", SDL_GetError());
        return 1;
    }

    SDL_Delay(3000);               // Chờ 3 giây
    SDL_DestroyWindow(window);     // Đóng và hủy bỏ cửa sổ
    SDL_Quit();                    // Kết thúc phiên làm việc SDL 2
    return 0;
}
```

Lưu ý: với các ứng dụng có dùng thư viện SDL 2, hàm `main` phải có dạng như sau:

```
int main(int argc, char* argv[]) { // Cho phép nhận tham số trên dòng lệnh
    // . . .
    return 0;
}
```

b. Đối tượng Renderer

Để thực hiện các thao tác dựng hình 2D với SDL 2, ta cần có một đối tượng quản lý việc dựng hình này trên các cửa sổ SDL đã được tạo ra. Đối tượng quản lý việc dựng hình này có tên gọi là `SDL_Renderer` và được liên kết với một cửa sổ thông qua con trỏ bởi đối tượng `SDL_Window`. Khi đó mọi thao tác dựng hình với đối tượng `SDL_Renderer` sẽ được kết xuất ra trên cửa sổ của đối tượng `SDL_Window`. Dưới đây là một số hàm API liên quan đến việc tạo và hủy đối tượn `SDL_Renderer`:

- `SDL_Renderer * SDL_CreateRenderer(SDL_Window * wnd, int index, Uint32 flags):`

  Tạo đối tượng quản lý ngữ cảnh dựng hình 2D cho một cửa sổ SDL. Ý nghĩa của các tham số được mô tả như sau:

- `wnd`: Con trỏ đến một cửa sổ SDL đã có.
- `index`: chỉ mục của driver dựng hình cần khởi tạo, thông thường là -1 để khởi tạo driver dựng hình đầu tiên mà SDL tìm thấy trong hệ thống.
- `flags`: cờ mô tả trạng thái ngữ cảnh dựng hình cần tạo, kết hợp với nhau bằng toán tử `|`. Gồm có một số giá trị như sau:
  - `SDL_RENDERER_SOFTWARE:` Sử dụng driver phần mềm để dựng hình. Chỉ sử dụng cho mục đích duy trì tính tương thích trên các hệ thống không có phần cứng tăng tốc đồ họa.
  - `SDL_RENDERER_ACCELERATED:` Sử dụng phần cứng tăng tốc đồ họa để dựng hình. Đây là giá trị thường được sử dụng nhất.
  - `SDL_RENDERER_PRESENTVSYNC:` Đồng bộ tốc độ frame với tốc độ làm tươi của màn hình thông qua cơ chế đồng bộ dọc (*Vertical Synchronization, V-Sync*). Với cơ chế đồng bộ này, tốc độ frame tối đa của ứng dụng sẽ xấp xỉ 50 hoặc 60 frame/giây tùy thuộc vào thiết bị hiển thị.

  Hàm này trả về con trỏ đến đối tượng ngữ cảnh dựng hình 2D `SDL_Renderer` nếu thành công. Ngược lại, hàm trả về một trị `NULL`.

- `void SDL_DestroyRenderer(SDL_Renderer * renderer)`

  Hàm này thực hiện việc hủy bỏ đối tượng ngữ cảnh dựng hình đã được tạo bởi hàm `SDL_CreateRenderer`. Các tài nguyên đã cấp phát cho đối tượng dựng hình này cũng được thu hồi.

c. Một số thao tác dựng hình cơ sở

Phần này trình bày một số thao tác dựng hình cơ sở với SDL 2. Ta cần lưu ý rằng SDL 2 dựng hình sử dụng cơ chế *double-buffering* (Hình 2). Mọi thao tác dựng hình sẽ kết xuất kết quả lên một vùng nhớ đệm (*framebuffer*). Nội dung vùng nhớ đệm này sẽ được hiển thị lên cửa sổ hoặc màn hình thông qua một lời gọi hàm sau khi ta đã hoàn thành việc dựng hình cho một frame.

- `int SDL_SetRenderDrawColor(SDL_Renderer * rend, Uint8 r, Uint8 g, Uint8 b, Uint8 a)`

  Thiết lập giá trị màu cho các thao tác dựng hình kế tiếp trên đối tượng `SDL_Renderer`. Giá trị màu được cho bởi bộ 4 `r`, `g`, `b` và `a`, lần lượt xác định cường độ của các thành

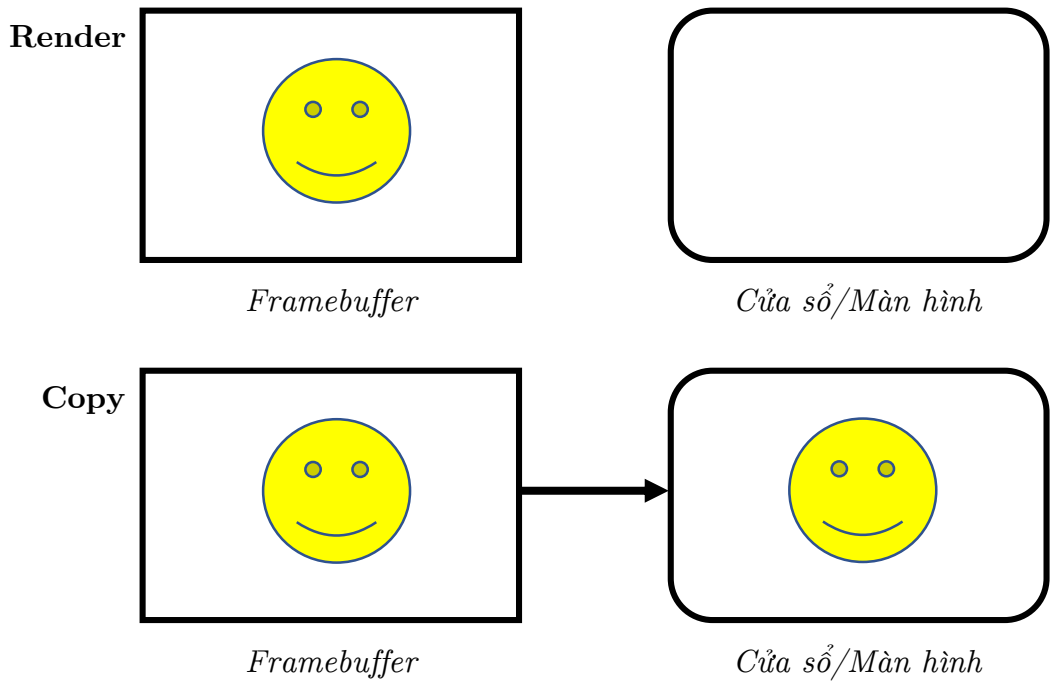
phần đỏ, xanh lá, xanh dương và alpha. Giá trị của tham số `a` thông thường được sử dụng là `SDL_ALPHA_OPAQUE`.

- `int SDL_RenderClear(SDL_Renderer * rend)`

  Xóa toàn bộ nội dung của framebuffer với giá trị màu đã được thiết lập bởi hàm `SDL_SetRenderDrawColor`.

- `void SDL_RenderPresent(SDL_Renderer * rend)`

  Tiến hành sao chép toàn bộ nội dung của vùng nhớ đệm dựng hình lên cửa sổ hoặc thiết bị hiển thị.



Hình 2. Cơ chế double-buffering

- `int SDL_RenderDrawPoint(SDL_Renderer * rend, int x, int y)`

  Vẽ một điểm ảnh lên vùng nhớ đệm tại tọa độ (x, y). Giá trị màu được thiết lập bởi hàm `SDL_SetRenderDrawColor`.

- `int SDL_RenderDrawLine(SDL_Renderer * rend, int x1, int y1, int x2, int y2)`

  Vẽ một đoạn thẳng bắt đầu từ điểm có tọa độ x_1, y_1 và kết thúc tại điểm có tọa độ (x_2, y_2), sử dụng màu được thiết lập bởi hàm `SDL_SetRenderDrawColor`.

- `int SDL_RenderDrawRect(SDL_Renderer * rend, const SDL_Rect * rect)`

  Vẽ một hình chữ nhật rỗng với tọa độ được cho trong biến `rect` có kiểu `SDL_Rect`. Kiểu dữ liệu `SDL_Rect` mô tả một hình chữ nhật trên vùng nhớ đệm thông qua các trường thông tin: (x, y, w, h). Tọa độ góc trên, trái của hình chữ nhật được xác định bởi tọa độ (x, y), chiều rộng và chiều cao lần lượt được xác định bởi w, h. Màu đường kẻ được xác định bởi hàm `SDL_SetRenderDrawColor`.

- `int SDL_RenderFillRect(SDL_Renderer * rend, const SDL_Rect * rect)`

  Tương tự hàm SDL_RenderDrawRect, tuy nhiên hàm này vẽ một hình chữ nhật được tô màu, với màu sắc được xác định bởi hàm `SDL_SetRenderDrawColor`.

d. Xử lý sự kiện

SDL 2 cho phép kiểm soát khá nhiều loại sự kiện xảy ra trên cửa sổ dựng hình. Các sự kiện này được phân loại dựa trên thiết bị phát sinh ra sự kiện đó, ví dụ như các sự kiện phát sinh từ bàn phím, chuột, joystick... Thông tin về các sự kiện xảy ra trên cửa sổ được mô tả bởi một cấu trúc dữ liệu có tên gọi `SDL_Event`. Phần này chỉ trình bày một số hàm xử lý các sự kiện nhận được từ bàn phím để kiểm soát hoạt động của ứng dụng mà thôi.

- `int SDL_PollEvent(SDL_Event * e)`

  Lấy một sự kiện ra khỏi hàng đợi sự kiện và lưu vào trong tham số `e` có kiểu dữ liệu `SDL_Event`. Nếu hàng đợi sự kiện rỗng, hàm này vẫn kết thúc và trả về trị 0, ngược lại, hàm sẽ trả về trị 1. Đây là hàm thường được chọn để sử dụng trong vòng lặp sự kiện của các ứng dụng viết trên nền SDL 2.

- `int SDL_PushEvent(SDL_Event * e)`

  Đưa một sự kiện mô tả bởi tham số `e` vào hàng đợi sự kiện.

- `SDL_WaitEvent(SDL_Event * e)`

  Chờ lấy sự kiện tiếp theo ra khỏi hàng đợi. Nếu hàng đợi sự kiện là rỗng, hàm sẽ ngưng hoạt động của ứng dụng để chờ cho đến khi xuất hiện một sự kiện trong hàng đợi để lấy ra.

Có 2 loại sự kiện được phát sinh từ bàn phím là `SDL_KEYDOWN` và `SDL_KEYUP`. Ta sử dụng các trường của cấu trúc dữ liệu `SDL_KeyboardEvent` để trích xuất thông tin từ các sự kiện này. `SDL_KeyboardEvent` thực chất là một `union` của cấu trúc dữ liệu `SDL_Event`.

Cấu trúc dữ liệu `SDL_KeyboardEvent` gồm có các trường sau:

- `Uint32 type`: mô tả loại sự kiện là `SDL_KEYDOWN` hoặc `SDL_KEYUP`.
- `Uint32 timestamp`: thời điểm xảy ra sự kiện.
- `Uint32 windowID`: ID của cửa sổ đang nhận sự kiện, nếu có.
- `Uint8 state`: trạng thái của phím là `SDL_PRESSED` hoặc `SDL_RELEASED`.
- `Uint8 repeat`: khác 0 nếu phím đang được nhấn lặp lại.
- `SDL_Keysym keysym`: Là một cấu trúc dữ liệu chứa các thông tin mô tả về phím đang xuất hiện bên trong sự kiện (`SDL_Keysym`).

Cấu trúc dữ liệu `SDL_Keysym` mô tả các thông tin về một phím khi được nhấn hoặc nhả trên bàn phím:

- `SDL_Scancode scancode`: là một `enum` mô tả mã phím vật lý. Chi tiết về `enum` này được trình bày tại địa chỉ `https://wiki.libsdl.org/SDL_Scancode`.
- `SDL_Keycode sym`: là một `enum` mô tả mã bàn phím ảo (có thể đã được ánh xạ lại). Chi tiết về `enum` này được trình bày tại địa chỉ: `https://wiki.libsdl.org/SDL_Keycode`.
- `Uint16 mod`: giá trị mô tả phím chức năng kèm theo (nếu có), là một tổ hợp của các giá trị sau đây: `KMOD_LSHIFT`, `KMOD_RSHIFT`, `KMOD_LCTRL`, `KMOD_RCTRL`, `KMOD_LALT`, `KMOD_RALT`, `KMOD_NUM`, `KMOD_CAPS`...

e. Xử lý va chạm đơn giản

- `SDL_bool SDL_HasIntersection(const SDL_Rect * a, const SDL_Rect * b)`

  Kiểm tra xem hình chữ nhật `a` có giao với hình chữ nhật `b` hay không. Nếu có chúng có va chạm.

Bài tập

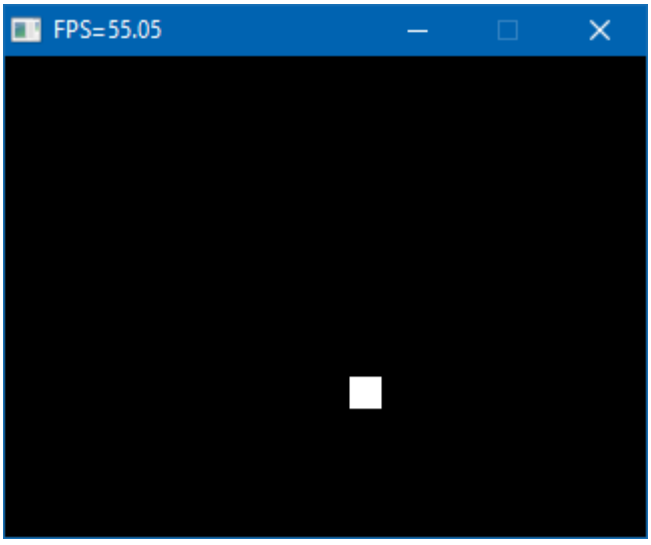
Bài 1: Bộ đếm frames-per-second

Xây dựng một ứng dụng đồ họa đơn giản sử dụng thư viện lập trình SDL 2 với các yêu cầu sau:

- Khởi tạo một cửa sổ đồ họa có kích thước 320×240. Cho phép bật hoặc tắt tính năng *V-Sync* thông qua tham số hàm khởi tạo cửa sổ.
- Cho một hình vuông kích thước 16×16 di chuyển bên trong cửa sổ. Khi hình vuông va chạm với các biên cửa sổ thì nảy ngược trở lại.
- Xây dựng một bộ đếm *frames-per-second*. Hiển thị tốc độ *frames-per-second* đo được hiện thời lên tiêu đề cửa sổ.

Lưu ý:

Thử nghiệm chương trình với tham số `vsync` của hàm `init` lần lượt là 0 và 1, đánh giá giá trị *frames-per-second* nhận được ở mỗi lần thử nghiệm. Trong đó, lời gọi hàm `init(0)` thực hiện khởi tạo hệ thống dựng hình SDL nhưng không đồng bộ tốc độ frame với chu kỳ làm tươi của màn hình. Ngược lại, lời gọi hàm `init(1)` kích hoạt việc đồng bộ hóa này.



Hình 3. Kết xuất của Bài 1 với V-Sync được bật

Tập tin `FPS.C`

```
#include "stdafx.h"
#include <stdio.h>
#include <SDL2/SDL.h>
```

```
#ifdef _MSC_VER                 // tham số liên kết với Visual C++
#   pragma comment (lib,    "sdl2main.lib")
#   pragma comment (lib,    "sdl2.lib")
#   pragma comment (linker, "/entry:\"mainCRTStartup\"" )
#   pragma comment (linker, "/subsystem:WINDOWS")
#endif

#define WWIDTH      320         // chiều rộng cửa sổ
#define WHEIGHT     240         // chiều cao cửa sổ
#define BLK_SIZE    16          // kích thước hình vuông

SDL_Window * wnd;               // con trỏ cửa sổ SDL
SDL_Renderer * rend;            // con trỏ đối tượng dựng hình
int x = WWIDTH/2, vx = 2;       // tọa độ và vận tốc di chuyển
int y = WHEIGHT/2, vy = 2;      // ban đầu của hình vuông

void block(int x, int y)        // vẽ hình vuông 16x16 tại tọa độ (x, y)
{                               // sử dụng màu trắng
    SDL_Rect r = {x, y, BLK_SIZE, BLK_SIZE};
    SDL_SetRenderDrawColor(rend, 255, 255, 255, 255 );
    SDL_RenderFillRect(rend, &r);
}

void init(int vsync)            // khởi tạo môi trường đồ họa SDL
{                               // vsync = 1 nếu muốn sử dụng V-Sync
    SDL_Init(SDL_INIT_VIDEO);   // khởi tạo hệ thống đồ họa

    // tạo cửa sổ tại vị trí (100, 100), kích thước 640x480.
    wnd = SDL_CreateWindow("FPS", 100, 100, WWIDTH, WHEIGHT, SDL_WINDOW_SHOWN);
    // tạo đối tượng dựng hình 2D, kiểm tra việc kích hoạt V-Sync
    rend = SDL_CreateRenderer(  wnd, -1,
                                SDL_RENDERER_ACCELERATED |
                                (vsync ? SDL_RENDERER_PRESENTVSYNC : 0));
}

int event()                     // xử lý sự kiện
{
    SDL_Event e;                // biến lưu trữ sự kiện
    int running = 1;            // biến kiểm soát vòng lặp
    while(SDL_PollEvent(&e)!=0) // lấy 1 sự kiện khỏi hàng đợi, nếu có
    {
        // nếu là phím nhấn, kiểm tra có phải ESC được nhấn hay không
        if(e.type == SDL_KEYDOWN)
            if (e.key.keysym.sym == SDLK_ESCAPE)
                running = 0;    // nếu đúng, ta yêu cầu đóng chương trình
        // nếu là sự kiện đóng cửa sổ (nhấn vào nút [X])
        if(e.type == SDL_QUIT )
            running = 0;        // ta cũng yêu cầu đóng chương trình
    }
    return running;
}

void update()                   // cập nhật trạng thái chương trình
```

```
{
    x += vx; y += vy;           // di chuyển hình vuông
    // nảy ngược lại nếu chạm các cạnh cửa sổ
    if (x < 0 || x > WWIDTH-BLK_SIZE)   vx = -vx;
    if (y < 0 || y > WHEIGHT-BLK_SIZE)  vy = -vy;
}

void draw()                     // xử lý các thao tác dựng hình
{
    // xóa toàn bộ framebuffer về màu đen
    SDL_SetRenderDrawColor(rend, 0, 0, 0, 0);
    SDL_RenderClear(rend);
    block(x, y);                // vẽ hình vuông tại vị trí hiện hành
    SDL_RenderPresent(rend);    // copy framebuffer -> screen
}

void done()                     // dọn dẹp ứng dụng
{
    SDL_DestroyRenderer(rend);  // hủy đối tượng ngữ cảnh dựng hình
    SDL_DestroyWindow(wnd);     // hủy cửa sổ SDL
    SDL_Quit();                 // chấm dứt phiên làm việc SDL
}

int main(int argc, char ** argv)
{
    int timerBegin, frameCount; // bộ đếm FPS
    float fps;                  // giá trị frame-per-second
    int running = 1;            // biến kiểm soát vòng lặp
    char buf[64];               // buffer tạm để hiển thị nội dung FPS

    init(1);                    // khởi tạo ứng dụng với VSync bật
    frameCount = 0;             // bắt đầu đếm số frame đã dựng
    timerBegin = SDL_GetTicks();// lấy thời gian bắt đầu vòng lặp
    while (running)
    {
        running = event();      // xử lý sự kiện trên cửa sổ
        update();               // cập nhật trạng thái ứng dụng
        draw();                 // dựng hình

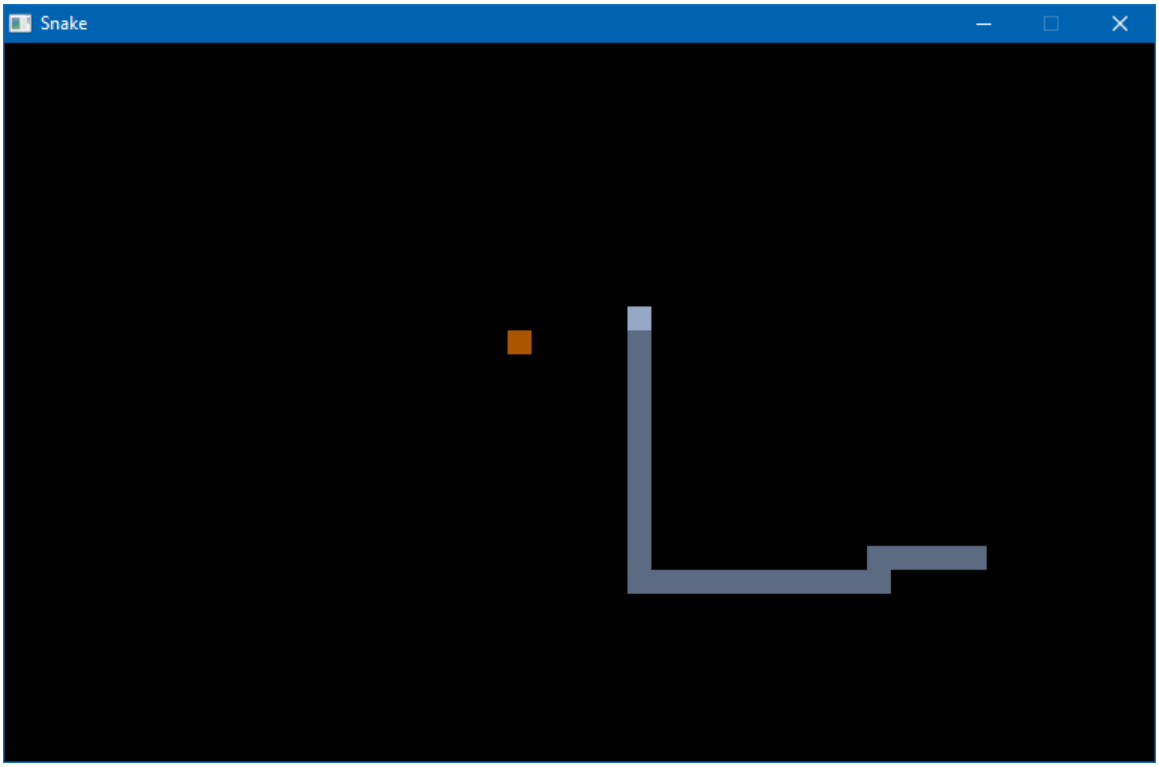
        frameCount++;           // đã dựng xong 1 frame

        // tính số FPS
        fps = frameCount / ((SDL_GetTicks() - timerBegin)/1000.0f);

        // và hiển thị lên tiêu đề cửa sổ
        sprintf(buf, "FPS=%0.2f", fps);
        SDL_SetWindowTitle(wnd, buf);
    }
    done();                     // kết thúc ứng dụng
    return 0;
}
```

Bài 2: Trò chơi Snake

Thiết kế và xây dựng trò chơi Rắn săn mồi (*Snake*). Với trò chơi này, người chơi điều khiển một đối tượng rắn di chuyển bên trong không gian trò chơi (Hình 4). Mồi được phát sinh ngẫu nhiên trong không gian trò chơi và nhiệm vụ của người chơi là hướng rắn đến "ăn" mồi. Việc "ăn" mồi sẽ làm chiều dài rắn tăng thêm một đơn vị. Quá trình này lặp đi lặp lại cho đến khi người chơi yêu cầu kết thúc trò chơi hoặc rắn va chạm với các cạnh của cửa sổ hoặc tự va chạm với chính mình.



Hình 4. Trò chơi Snake hoàn thiện

Yêu cầu:

- Cài đặt trò chơi *Snake* sử dụng ngôn ngữ C hoặc C++, kết hợp thư viện đồ họa SDL 2. Cơ chế hoạt động của trò chơi là di chuyển con rắn trong không gian trò chơi để "ăn" mồi.
- Trạng thái ban đầu:
    - Con rắn có vị trí ban đầu nằm chính giữa cửa sổ trò chơi, độ dài ban đầu 2 đơn vị.
    - Mồi được phát sinh ngẫu nhiên trên không gian trò chơi nhưng không được trùng với vị trí của con rắn.
- Điều khiển: Người chơi sử dụng các phím $\uparrow, \downarrow, \leftarrow, \rightarrow$ để di chuyển con rắn. Rắn có thể di chuyển theo bốn hướng lên, xuống, trái, phải ứng với các phím điều khiển. Tuy nhiên, con rắn không thể di chuyển lùi.
- Xử lý va chạm đơn giản giữa con rắn và mồi.
- Xử lý va chạm giữa đầu rắn và thân rắn.

- Cho phép rắn khi di chuyển đến các cạnh màn hình thì xuất hiện ở cạnh thuộc phía đối diện.
- Để đơn giản, con rắn được thể hiện là một loạt các ô vuông cùng kích thước, liên tục nhau. Mồi là một hình vuông.

Phân tích:

- *Không gian trò chơi*: được chia thành một lưới gồm có m ô theo chiều ngang và n ô theo chiều dọc. Mỗi ô có kích thước cố định. Ta chọn kích thước không gian trò chơi là 48×30 ô, mỗi ô có kích thước 16×16 *pixel*.
- *Con rắn*: Con rắn được tổ chức là một mảng các cặp tọa độ (x, y) và một giá trị cho biết chiều dài hiện tại của rắn.
- *Mồi:* Là một ô có tọa độ (x, y) trên không gian trò chơi và có thêm một giá trị màu ngẫu nhiên.
- *Xử lý va chạm*: Trò chơi này chỉ có 2 loại va chạm
    - Va chạm giữa đầu rắn và mồi: nếu đầu rắn có cùng tọa độ với mồi, ta xem như rắn đã "ăn" mồi. Khi đó, độ dài rắn được tăng thêm một đơn vị, mồi lại được phát sinh ngẫu nhiên trong không gian trò chơi với yêu cầu không được trùng với vị trí của con rắn.
    - Đối với trường hợp đầu rắn va chạm với một trong các cạnh của cửa sổ, ta cho phép rắn xuất hiện ở cạnh đối diện. Điều này làm giảm độ khó của trò chơi và giảm sự phức tạp khi cài đặt các bước xử lý va chạm.
    - Va chạm giữa đầu rắn và thân rắn: Ta kết thúc trò chơi.

Hướng phát triển:

- Sử dụng *sprite* để biểu diễn con rắn và mồi.
- Cho phép tạm dừng trò chơi.
- Cho phép người chơi có số lượng tối đa n mạng (*life*). Mỗi lần rắn và chạm với chính nó, người chơi mất đi một mạng. Trò chơi kết thúc khi người chơi không còn mạng nào.
- Hiển thị số lượng mạng còn lại của người chơi. Ngoài ra, cho phép tính và hiển thị điểm số của người chơi mỗi khi rắn "ăn" mồi.

Mã nguồn:

- Tập tin `GLOBAL.H`

```
#pragma once
#include <SDL2/sdl.h>
#include <stdlib.h>
#include <time.h>

#define BOARD_CELL          16  /* kích thước của 1 ô, nên là lũy thừa của 2 */
#define BOARD_HORZ          48  /* số lượng ô theo chiều ngang */
#define BOARD_VERT          30  /* số lượng ô theo chiều dọc */

/* 18 màu sắc có thể sử dụng được trong trò chơi */
#define CR_BLACK        0
#define CR_BLUE         1
```

```
#define CR_GREEN        2
#define CR_CYAN         3
#define CR_RED          4
#define CR_MAGENTA      5
#define CR_BROWN        6
#define CR_GRAY         7
#define CR_DKGRAY       8
#define CR_BRBLUE       9
#define CR_BRGREEN      10
#define CR_BRCYAN       11
#define CR_BRRED        12
#define CR_BRMAGENTA    13
#define CR_YELLOW       14
#define CR_WHITE        15
#define CR_SNAKE_HEAD   16
#define CR_SNAKE_BODY   17

/* tổng số màu có thể sử dụng, bao gồm 16 màu EGA chuẩn và 2 màu dành cho rắn */
#define CR_MAX_COLORS   18

/* các hướng di chuyển có thể */
typedef enum direction {DOWN, LEFT, RIGHT, UP} DIRECTION;

/* cấu trúc lưu trữ tọa độ 2D */
typedef struct coord {
    int x, y;
} COORD;

/* các biến ngoại biên */
extern SDL_Window * g_window;       /* SDL window */
extern SDL_Renderer * g_renderer;   /* SDL renderer */

/* bảng giá trị các màu sắc có thể sử dụng được */
extern unsigned long GAME_PALETTE[CR_MAX_COLORS];

/* thiết lập giá trị màu hiện thời, sử dụng bảng màu đã cho */
void setColor(int c);

/* vẽ một hình chữ nhật */
void rectDraw(int x, int y, int w, int h, unsigned long color);
```

- Tập tin `GLOBAL.CPP`

```
#include "stdafx.h"
#include "global.h"

SDL_Window * g_window;
SDL_Renderer * g_renderer;

/* liệt kê các giá trị màu */
unsigned long  GAME_PALETTE[CR_MAX_COLORS] =
    {0x000000, 0x0000AA, 0x00AA00, 0x00AAAA,
     0xAA0000, 0xAA00AA, 0xAA5500, 0xAAAAAA,
```

```
     0x555555, 0x5555FF, 0x55FF55, 0x55FFFF,
     0xFF5555, 0xFF55FF, 0xFFFF55, 0xFFFFFF,
     0x95A7C4, 0x5D6B82};

void setColor(int c)
{
    Uint32 color = GAME_PALETTE[c];
    SDL_SetRenderDrawColor( g_renderer,
                            (color >> 16),       /* tách thành phần R */
                            (color >> 8) & 0xFF, /* tách thành phần G */
                            color & 0xFF,        /* tách thành phần B */
                             0xFF);
}

void rectDraw(int x, int y, int w, int h, unsigned long color)
{
    SDL_Rect    rc = {x, y, w, h};
    setColor(color);
    SDL_RenderFillRect(g_renderer, &rc);
}
```

- Tập tin `ENTITIES.H`

```
#pragma once
#include "global.h"

#define SNAKE_MAXLEN    100 /* chiều dài tối đa của rắn */
#define SNAKE_INITLEN   2   /* chiều dài ban đầu của rắn */

typedef struct snake {
    COORD   coords[SNAKE_MAXLEN];   /* rắn là một danh sách các tọa độ 2D */
    int     len;                    /* độ dài hiện thời của rắn */
} SNAKE;

typedef struct fruit {
    COORD   loc;                    /* vị trí của mồi */
    Uint32  color;                  /* màu sắc */
} FRUIT;

/* vẽ một ô của thân rắn hoặc đầu rắn */
void snakeCell(COORD c, SDL_bool head);

/* khởi tạo rắn */
void snakeInit(SNAKE * s);

/* vẽ rắn */
void snakeDraw(const SNAKE * s);

/* cập nhật trạng thái của rắn */
void snakeUpdate(SNAKE * s);

/* phát sinh mồi */
void fruitGen(FRUIT * f, const SNAKE * s);
```

```
/* vẽ mồi */
void fruitDraw(const FRUIT * f);
```

- Tập tin ENTITIES.CPP

```
#include "stdafx.h"
#include "entities.h"

void snakeCell(COORD c, SDL_bool head)
{
    int color = head ? CR_SNAKE_HEAD : CR_SNAKE_BODY;
    rectDraw(c.x * BOARD_CELL, c.y * BOARD_CELL, BOARD_CELL, BOARD_CELL, color);
}

void snakeInit(SNAKE * s)
{
    s->coords[0].x = (BOARD_HORZ >> 1);
    s->coords[0].y = (BOARD_VERT >> 1);
    s->len = SNAKE_INITLEN;
    for (int i = 1; i < s->len; i++) {
        s->coords[i].x = s->coords[i-1].x;
        s->coords[i].y = (BOARD_VERT >> 1)+i;
    }
}

void snakeDraw(const SNAKE * s)
{
    snakeCell(s->coords[0], SDL_TRUE);
    for(int i = 1; i < s->len; i++)
        snakeCell(s->coords[i], SDL_FALSE);
}

void snakeUpdate(SNAKE * s)
{
    for (int i = s->len; i > 0; i--)
        s->coords[i] = s->coords[i-1];
}

void fruitGen(FRUIT * f, const SNAKE * s)
{
    SDL_bool ok = SDL_FALSE;
    int i;
    while (!ok) {
        f->loc.x = rand() % BOARD_HORZ;
        f->loc.y = rand() % BOARD_VERT;
        for(i = 0; i < s->len;i++) {
            if (f->loc.x != s->coords[i].x &&
                f->loc.y != s->coords[i].y)
                continue;
        }
        if (i == s->len)
            ok = SDL_TRUE;
```

```
    }
    f->color = rand() % 14 + 1;
}

void fruitDraw(const FRUIT * f)
{
    rectDraw(f->loc.x * BOARD_CELL, f->loc.y * BOARD_CELL,
             BOARD_CELL, BOARD_CELL, f->color);
}
```

- Tập tin GAME.H

```
#pragma once
#include "stdafx.h"
#include "global.h"
#include "game.h"

#define GAME_MAX_FPS        12  /* giới hạn tốc độ khung đến 12fps */
#define GAME_FRAME_TIME     (1000 / GAME_MAX_FPS)

typedef struct game         /* cấu trúc trò chơi bao gồm */
{
    FRUIT f;                /* mồi */
    SNAKE s;                /* rắn */
    DIRECTION dir;          /* hướng di chuyển của rắ n*/
    SDL_bool running;       /* cờ trạng thái hoạt động */
} GAME;

/* khởi tạo trò chơi */
void gameInit(GAME * g, SDL_bool fullscreen);

/* kết thúc trò chơi */
void gameShutdown(GAME * g);

/* vòng lặp trò chơi */
void gameLoop(GAME * g);

/* cập nhật trạng thái trò chơi */
void gameUpdate(GAME * g);

/* dựng hình cho 1 frame */
void gameDraw(GAME * g);

/* xử lý input */
void gameInput(GAME * g);
```

- Tập tin GAME.CPP

```
#include "stdafx.h"
#include "game.h"

void gameInit(GAME * g, SDL_bool fullscreen)
{
```

```
    srand(time(0));
    SDL_Init(SDL_INIT_VIDEO);
    g_window = SDL_CreateWindow("Snake",
                                SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                                BOARD_CELL * BOARD_HORZ, BOARD_CELL * BOARD_VERT,
                                SDL_WINDOW_SHOWN |
                               (fullscreen ? SDL_WINDOW_FULLSCREEN : 0));
    g_renderer = SDL_CreateRenderer(g_window, -1,
                                    SDL_RENDERER_ACCELERATED |
                                    SDL_RENDERER_PRESENTVSYNC);
    g->dir = UP;
    snakeInit(&g->s);
    fruitGen(&g->f, &g->s);
}

void gameShutdown(GAME * g)
{
    SDL_DestroyRenderer(g_renderer);
    SDL_DestroyWindow(g_window);
    SDL_Quit();
}

void gameUpdate(GAME * g)
{
    snakeUpdate(&g->s);         /* cập nhật trạng thái của rắn */
    switch(g->dir) {            /* di chuyển rắn */
    case UP:    g->s.coords[0].y--; break;      /* đi lên */
    case DOWN:  g->s.coords[0].y++; break;      /* đi xuống */
    case LEFT:  g->s.coords[0].x--; break;      /* sang trái */
    case RIGHT: g->s.coords[0].x++; break;      /* sang phải */
    }

    /* rắn đã ăn mồi */
    if (g->s.coords[0].x == g->f.loc.x &&
        g->s.coords[0].y == g->f.loc.y) {
        g->s.len++;             /* thân rắn dài thêm 1 đơn vị */
        fruitGen(&g->f, &g->s); /* phát sinh lại mồi tại vị trí khác */
    }

    /* cho phép rắn đi xuyên qua các biên */
    if (g->s.coords[0].x >= BOARD_HORZ) g->s.coords[0].x = 0;
    if (g->s.coords[0].x < 0) g->s.coords[0].x = BOARD_HORZ;
    if (g->s.coords[0].y >= BOARD_VERT) g->s.coords[0].y = 0;
    if (g->s.coords[0].y < 0) g->s.coords[0].y = BOARD_VERT;

    /* kiểm tra va chạm giữa đầu và thân rắn, nếu có va chạm: kết thúc trò chơi */
    for (int i = 1; i < g->s.len; i++)
        if (g->s.coords[0].x == g->s.coords[i].x &&
            g->s.coords[0].y==g->s.coords[i].y)
            g->running = SDL_FALSE;
}

void gameDraw(GAME * g)
{
```

```
    SDL_SetRenderDrawColor(g_renderer, 0, 0, 0, 0);
    SDL_RenderClear(g_renderer);
    fruitDraw(&g->f);   /* vẽ mồi */
    snakeDraw(&g->s);   /* vẽ rắn */
    SDL_RenderPresent(g_renderer);
}

void gameInput(GAME * g)
{
    SDL_Event   e;
    SDL_PollEvent(&e);
    switch (e.type)
    {
    case SDL_KEYDOWN:
        switch(e.key.keysym.sym)   /* xác định hướng di chuyển, chống đi lùi */
        {
        case SDLK_UP:       if (g->dir != DOWN) g->dir = UP; break;
        case SDLK_DOWN:     if (g->dir != UP) g->dir = DOWN; break;
        case SDLK_LEFT:     if (g->dir != RIGHT) g->dir = LEFT; break;
        case SDLK_RIGHT:    if (g->dir != LEFT) g->dir = RIGHT; break;
        case SDLK_ESCAPE:   g->running = SDL_FALSE; break;
        }
        break;
    case SDL_QUIT:
        g->running = SDL_FALSE;
        break;
    }
}

void gameLoop(GAME * g)
{
    long timerBegin, timerDiff, timerSleep;
    g->running = SDL_TRUE;
    while (g->running) {
        timerBegin = SDL_GetTicks();
        gameUpdate(g);            /* cập nhật trạng thái trò chơi */
        gameInput(g);             /* xử lý input */
        gameDraw(g);              /* dựng một frame hình */
        /* tính toán thời gian dựng xong 1 frame hình để giới hạn frame rate */
        timerDiff = SDL_GetTicks() - timerBegin;
        timerSleep = GAME_FRAME_TIME - timerDiff;
        if(timerSleep > 0)
            SDL_Delay(timerSleep);/* chờ cho đủ thời gian 12fps */
    }
}
```

- Tập tin `SNAKE.CPP`

```
#include "game.h"

#ifdef _MSC_VER         /* chỉ thị liên kết với Visual C++ */
#   pragma comment (lib,    "sdl2main.lib")
#   pragma comment (lib,    "sdl2.lib")
```

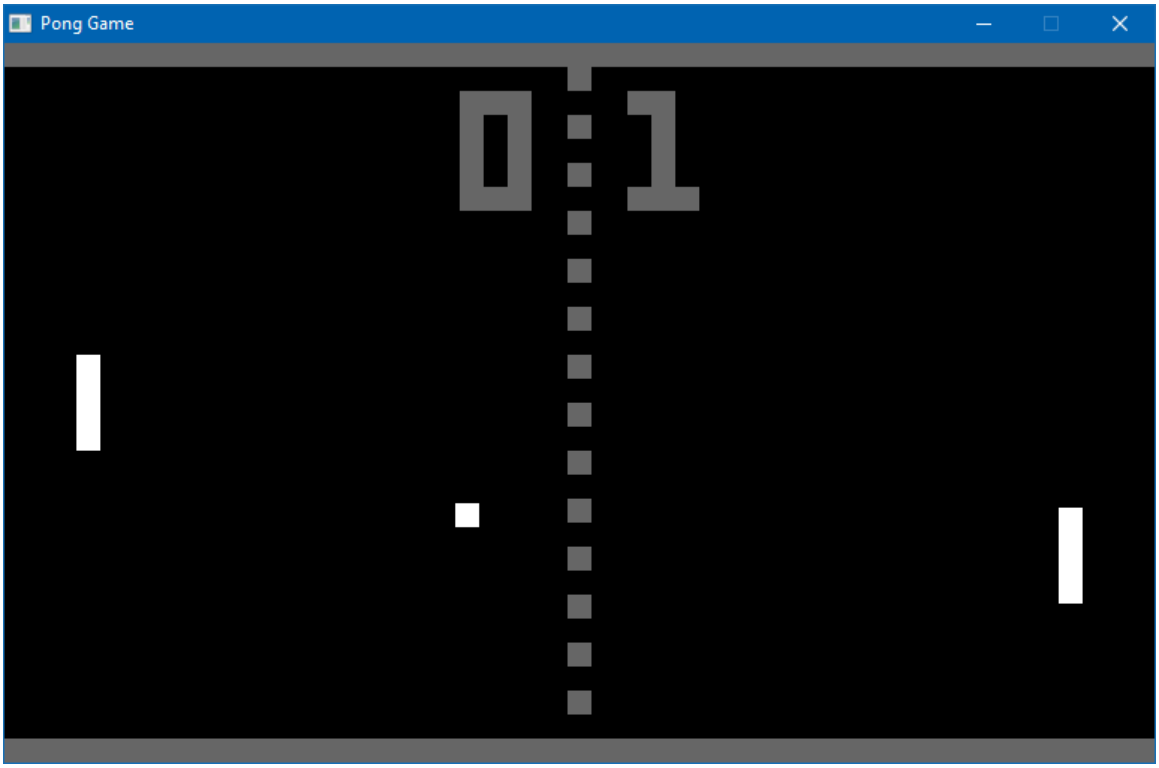
```
#   pragma comment (linker, "/entry:\"mainCRTStartup\"" )
#   pragma comment (linker, "/subsystem:WINDOWS")
#endif

int main(int argc, char ** argv)
{
    GAME g;
    gameInit(&g, SDL_FALSE);
    gameLoop(&g);
    gameShutdown(&g);
    return 0;
}
```

Bài 3: Trò chơi Pong

Thiết kế và xây dựng trò chơi Bóng bàn, *Pong* (https://en.wikipedia.org/wiki/Pong).

Đây là trò chơi đối kháng giữa 2 người chơi hoặc giữa một người chơi với máy tính (Hình 5). Nhiệm vụ của người chơi là sử dụng một ván (*paddle*) di chuyển lên hoặc xuống để hứng quả bóng di chuyển trên màn hình. Khi quả bóng va chạm với ván của người chơi hoặc 2 đáy của cửa sổ sẽ nảy ngược trở lại. Nếu một người chơi hứng trượt quả bóng, 1 điểm sẽ được tính cho người chơi còn lại. Trò chơi sẽ kết thúc khi một trong hai người chơi đạt đến điểm 10 hoặc người chơi yêu cầu kết thúc sớm trò chơi.



Hình 5. Trò chơi Pong hoàn thiện

Yêu cầu:

- Cài đặt trò chơi Pong sử dụng ngôn ngữ C hoặc C++, kết hợp thư viện đồ họa SDL 2. Cơ chế hoạt động của trò chơi là đối kháng giữa 2 người chơi. Có hoặc không sử dụng V-Sync, cho phép sử dụng cửa sổ toàn màn hình nếu cần.

- Trạng thái ban đầu của trò chơi:
    - Bóng xuất hiện chính giữa cửa sổ. Hướng di chuyển ban đầu của quả bóng theo góc 45^o ngẫu nhiên về hai phía của màn hình.
    - Ván của người chơi thứ nhất phía bên trái, ở giữa màn hình theo chiều dọc.
    - Ván của người chơi thứ hai phía bên phải, ở giữa màn hình theo chiều dọc.
- Điều khiển:
    - Ván của người chơi thứ nhất được điều khiển bởi phím W (di chuyển lên) và S (di chuyển xuống).
    - Tương tự, ván của người chơi thứ hai được điều khiển bởi phím $\uparrow$ và $\downarrow$ trên bàn phím.
- Điểm số của hai người chơi được hiển thị trên màn hình với font chữ được định nghĩa bởi 10 ma trận nhị phân kích thước 5×3, biểu diễn các ký số từ 0 đến 9.
- Xử lý va chạm đơn giản giữa bóng với 2 đáy của cửa sổ và giữa bóng với 2 ván của người chơi.
- Để đơn giản, bóng và ván của người chơi được thể hiện dưới dạng hình chữ nhật hoặc hình vuông. Sinh viên có thể mở rộng trò chơi bằng cách sử dụng *sprite* để đại diện cho quả bóng và ván của người chơi.

Phân tích

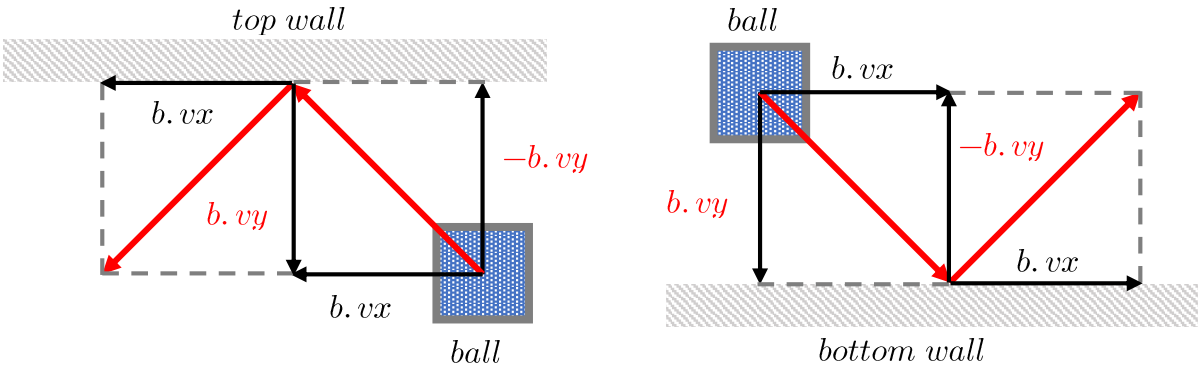
a. *Cửa sổ trò chơi*: Ta chia cửa sổ trò chơi thành một lưới gồm các ô vuông (*cell*) có kích thước 16×16 để tiện cho việc hiển thị và tính toán tọa độ các đối tượng. Kích thước cửa sổ trò chơi trong bài tập này là 48×30 $cell = 768 \times 480$ $pixel$. Ta có thể điều chỉnh kích thước này bằng cách thay đổi số lượng ô theo chiều ngang và số lượng ô theo chiều dọc của cửa sổ.

b. *Quả bóng*: để biểu diễn quả bóng trên màn hình, ta cần các thông tin sau:
    - Tọa độ (x, y) của quả bóng.
    - Vector vx và vy biểu diễn hướng di chuyển lần lượt theo 2 trục x và y.
    - Để đơn giản, ta xem quả bóng là một hình vuông, kích thước 1×1 *cell*.

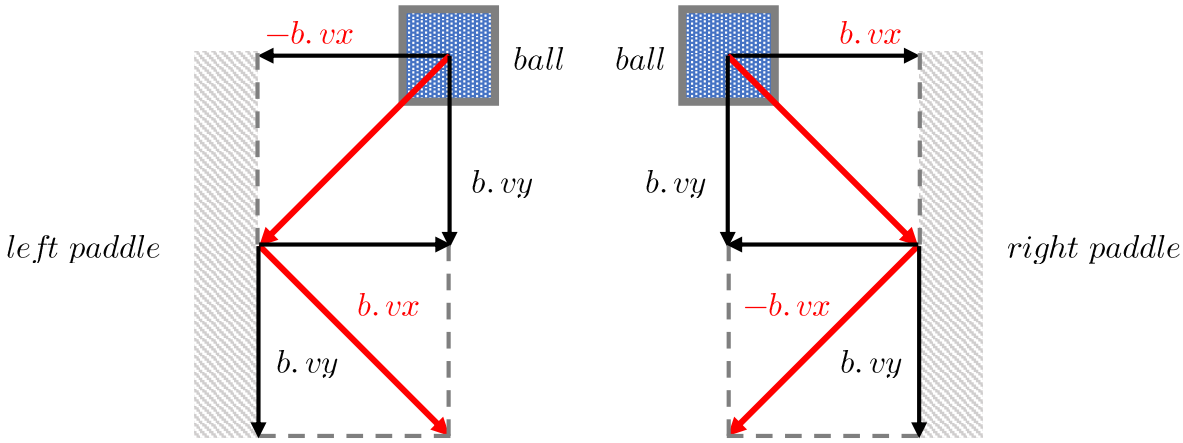
c. *Ván của người chơi*: tương tự, để biểu diễn ván của người chơi trên màn hình, ta cần các thông tin sau:
    - Tọa độ (x, y) của ván.
    - Hướng di chuyển của ván là *UP*, *DOWN* hoặc *STOP*.
    - Tốc độ di chuyển lên, xuống của ván.
    - Điểm số của người chơi đang sử dụng ván.
    - Hình dạng ván là một hình chữ nhật, kích thước 1×5 *cell*.

d. *Xử lý va chạm*: ta chỉ xử lý 2 loại va chạm đơn giản trong trò chơi *Pong*, bao gồm:
    - Bóng va chạm với đỉnh, đáy của cửa sổ: ta đổi hướng vector vy của quả bóng để nảy ngược quả bóng lại theo trục y. Vector vx được giữ nguyên để bảo toàn hướng chuyển động theo trục x (Hình 6).
    - Va chạm giữa bóng và mặt trong của 2 ván đại diện cho 2 người chơi: tương tự, ta đổi hướng vector vx của quả bóng và giữ nguyên vector vy để tạo cảm giác quả bóng nảy ngược lại theo trục x (Hình 7). Để kiểm tra va chạm giữa

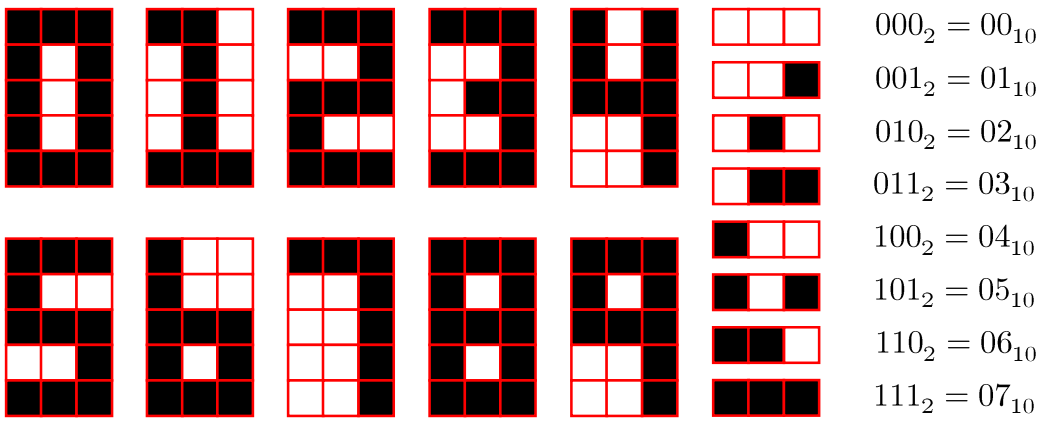
quả bóng và ván của người chơi, hình chữ nhật bao quanh 2 đối tượng này có giao nhau hay không thông qua hàm `SDL_HasIntersection`.



Hình 6. Xử lý va chạm giữa bóng và hai đáy của cửa sổ



Hình 7. Xử lý va chạm giữa quả bóng và ván của người chơi



Hình 8. Font chữ biểu diễn các ký số và các giá trị nhị phân biểu diễn các pixel trong từng ký số.

e. *Hiển thị điểm số*: Để hiển thị điểm số của người chơi, ta sử dụng bộ font chữ gồm 10 ký tự, mỗi ký tự có kích thước 5 dòng, 3 cột (5×3).

Ta có thể sử dụng mười mảng 1 chiều gồm có 5 phần tử gồm các số nguyên 8-bit, hoặc một mảng 1 chiều có kích thước $10 \times 5 = 50$ số nguyên 8-bit. Mỗi bộ 5 số nguyên biểu diễn 5 dòng của một ký số ở dạng nhị phân (Hình 8).

Để vẽ một ký số lên màn hình, ta sử dụng vòng lặp duyệt qua các bit 1 của các bộ 5 số nguyên này và hiển thị chúng lên màn hình, mỗi điểm ảnh nhị phân được hiển thị với kích thước của một ô 16×16 *pixel*.

Hướng phát triển

Trò chơi Pong được trình bày trong phần này còn khá đơn giản, chỉ đáp ứng được các yêu cầu cơ bản đặt ra trong đề bài. Để hoàn thiện trò chơi, ta có thể bổ sung một số yêu cầu như sau:

- Cho phép người chơi chọn chế độ chơi thông qua một menu đơn giản.
- Hỗ trợ chế độ đối kháng giữa người với máy tính.
- Cho phép tạm dừng trò chơi.
- Xử lý đầy đủ các trường hợp va chạm giữa bóng và ván của người chơi.
- Cho phép quả bóng di chuyển với các góc di chuyển và tốc độ di chuyển khác nhau phụ thuộc vào điểm va chạm giữa bóng và ván.
- Sử dụng hình ảnh để đại diện cho đối tượng quả bóng và ván dưới dạng *sprite*.

Mã nguồn trò chơi

- Tập tin `GLOBAL.H`

```
#pragma once
#include <SDL2/SDL.h>
#include <stdlib.h>
#include <time.h>

#define BOARD_CELL      16        /* kích thước 1 ô, nên là lũy thừa của 2 */
#define BOARD_WIDTH     (BOARD_CELL * 48)          /* chiều rộng cửa sổ */
#define BOARD_HEIGHT    (BOARD_CELL * 30)          /* chiều cao cửa sổ */
#define BOARD_TOP       BOARD_CELL                 /* biên đỉnh */
#define BOARD_BOTTOM    (BOARD_HEIGHT-BOARD_CELL)  /* biên đáy */

#define CR_WHITE        0xFF            /* màu trắng */
#define CR_GRAY         0x66            /* màu xám */
#define CR_BLACK        0x00            /* màu đen */

#define DIGIT_WIDTH     3               /* chiều rộng 1 ký số */
#define DIGIT_HEIGHT    5               /* chiều cao 1 ký số */

/* khai báo các biến toàn cục */
extern SDL_Window * g_window;           /* SDL window */
extern SDL_Renderer * g_renderer;       /* SDL renderer */
extern char DIGITS[10*DIGIT_HEIGHT];    /* font các ký số */

/* phát sinh vector chuyển động ngẫu nhiên */
int velocity();

/* kiểm tra va chạm giữa hai SDL_Rect */
SDL_bool rectCollide(const SDL_Rect * a, const SDL_Rect * b);

/* vẽ một hình chữ nhật */
void rectDraw(int x, int y, int w, int h, int cr);
```

```
/* vẽ một ký số sử dụng font chữ đã được định nghĩa */
void digitDraw(int x, int y, int digit);
```

- Tập tin GLOBAL.CPP

```
#include "stdafx.h"
#include "global.h"

SDL_Window * g_window = NULL;           /* SDL window */
SDL_Renderer * g_renderer = NULL;       /* SDL renderer */
char DIGITS[10*DIGIT_HEIGHT] =          /* font chữ cho các ký số */
{
    0x07, 0x05, 0x05, 0x05, 0x07,   /* 0 */
    0x06, 0x02, 0x02, 0x02, 0x07,   /* 1 */
    0x07, 0x01, 0x07, 0x04, 0x07,   /* 2 */
    0x07, 0x01, 0x03, 0x01, 0x07,   /* 3 */
    0x05, 0x05, 0x07, 0x01, 0x01,   /* 4 */
    0x07, 0x04, 0x07, 0x01, 0x07,   /* 5 */
    0x07, 0x04, 0x07, 0x05, 0x07,   /* 6 */
    0x07, 0x01, 0x01, 0x01, 0x01,   /* 7 */
    0x07, 0x05, 0x07, 0x05, 0x07,   /* 8 */
    0x07, 0x05, 0x07, 0x01, 0x01    /* 9 */
};

int velocity()
{
    int value = 0;   /* phát sinh ngẫu nhiên một trong hai giá trị là -1 hoặc 1 */
    do { value = 1-(rand() % 3); } while (!value);
    return value;
}

SDL_bool rectCollide(const SDL_Rect * a, const SDL_Rect * b)
{
    /* kiểm tra hình chữ nhật a, b có giao nhau hay không */
    return SDL_HasIntersection(a, b);
}

void rectDraw(int x, int y, int w, int h, int cr)
{
    /* vẽ hình chữ nhật, đây là hàm được sử dụng để vẽ mọi đối tượng trong trò chơi */
    SDL_Rect rc = {x, y, w, h};
    SDL_SetRenderDrawColor(g_renderer, cr, cr, cr, 0xFF);
    SDL_RenderFillRect(g_renderer, &rc);
}

void digitDraw(int x, int y, int digit)                 /* vẽ một ký số */
{
    for (int i = 0; i < DIGIT_HEIGHT; i++) {            /* lặp 5 dòng */
        char value = DIGITS[digit*DIGIT_HEIGHT+i];
        for (int j = 0; j < DIGIT_WIDTH; j++) {         /* lặp 3 cột */
            if (value & 0x4)                            /* kiểm tra bit thứ 3 */
                rectDraw(x+j*BOARD_CELL,                /* vẽ 1 ô nếu bit=1 */
```

```
                        y+i*BOARD_CELL,
                        BOARD_CELL, BOARD_CELL, CR_GRAY);
            value <<= 1;                                /* dịch trái 1 bit */
        }
    }
}
```

- Tập tin BALL.H

```
#pragma once
#include "global.h"

#define BALL_SIZE           BOARD_CELL    /* kích thước quả bóng */
#define BALL_SPEED          4             /* tốc độ di chuyển */
#define BALL_COLLIDE_TOP    (BOARD_TOP)   /* biên va chạm */
#define BALL_COLLIDE_BOTTOM (BOARD_BOTTOM-BALL_SIZE)
#define BALL_START_X        (BOARD_WIDTH-BALL_SIZE) >> 1      /* tọa độ ban đầu */
#define BALL_START_Y        (BOARD_HEIGHT-BALL_SIZE) >> 1

typedef struct ball  {
    int x, y;       /* tọa độ quả bóng */
    int vx, vy;     /* vector chuyển động */
} BALL;

/* khởi tạo trạng thái ban đầu của quả bóng*/
void ballInit(BALL * ball, int x, int y);

/* vẽ quả bóng */
void ballDraw(BALL * ball);

/* cập nhật vị trí quả bóng */
void ballUpdate(BALL * ball);
```

- Tập tin BALL.CPP

```
#include "stdafx.h"
#include "ball.h"

void ballInit(BALL * ball, int x, int y)
{
    ball->x = x;
    ball->y = y;
    ball->vx = velocity();  /* phát sinh ngẫu nhiên vector chuyển động */
    ball->vy = velocity();
}

void ballDraw(BALL * ball)
{
    rectDraw(ball->x, ball->y, BALL_SIZE, BALL_SIZE, CR_WHITE);
}

void ballUpdate(BALL * ball)
{
    ball->x += BALL_SPEED * ball->vx;
```

```
    ball->y += BALL_SPEED * ball->vy;
}
```

- Tập tin PADDLE.H

```
#pragma once
#include "global.h"

#define PADDLE_WIDTH            BOARD_CELL      /* chiều rộng ván */
#define PADDLE_HEIGHT           BOARD_CELL*4    /* chiều cao ván */
#define PADDLE_SPEED            16              /* tốc độ di chuyển */
#define PADDLE_GAP              BOARD_CELL*3    /* khoảng cách đến biên màn hình */
#define PADDLE_LEFT             (PADDLE_GAP)
#define PADDLE_RIGHT            (BOARD_WIDTH-PADDLE_WIDTH-PADDLE_GAP)
#define PADDLE_COLLIDE_TOP      (BOARD_TOP)     /* biên va chạm */
#define PADDLE_COLLIDE_BOTTOM   (BOARD_BOTTOM-PADDLE_HEIGHT)

/* enum mô tả hướng di chuyển */
typedef enum paddle_dir {PADDLE_STOP, PADDLE_UP, PADDLE_DOWN} PAD_DIR;
typedef struct paddle {
    int x, y;       /* tọa độ ván */
    int score;      /* điểm số */
    PAD_DIR dir;    /* hướng di chuyển */
} PADDLE;

/* khởi tạo cấu trúc dữ liệu mô tả ván */
void paddleInit(PADDLE * pad, int x, int y);

/* vẽ tấm ván */
void paddleDraw(PADDLE * pad);

/* cập nhật vị trí của tấm ván */
void paddleUpdate(PADDLE * pad);
```

- Tập tin PADDLE.CPP

```
#include "stdafx.h"
#include "paddle.h"

void paddleInit(PADDLE * pad, int x, int y)
{
    pad->x = x;
    pad->y = y;
    pad->score = 0;         /* điểm số ban đầu */
    pad->dir = PADDLE_STOP; /* hướng di chuyển ban đầu */
}

void paddleDraw(PADDLE * pad)
{
    rectDraw(pad->x, pad->y, PADDLE_WIDTH, PADDLE_HEIGHT, CR_WHITE);
}

void paddleUpdate(PADDLE * pad)
```

```
{
    if (pad->dir == PADDLE_UP) {   /* di chuyển lên, có kiểm tra va chạm */
        if (pad->y - PADDLE_SPEED > PADDLE_COLLIDE_TOP)
            pad->y -= PADDLE_SPEED;
        else
            pad->y = PADDLE_COLLIDE_TOP;
    }
    else
    if (pad->dir == PADDLE_DOWN) { /* di chuyển xuống, có kiểm tra va chạm */
        if (pad->y + PADDLE_SPEED < PADDLE_COLLIDE_BOTTOM)
            pad->y += PADDLE_SPEED;
        else
            pad->y = PADDLE_COLLIDE_BOTTOM;
    }
    pad->dir = PADDLE_STOP;        /* dừng chuyển động */
}
```

- Tập tin `GAME.H`

```
#pragma once
#include "global.h"
#include "ball.h"
#include "paddle.h"

#define GAME_MAXSCORE   10  /* điểm số tối đa */
#define GAME_MAX_FPS    50  /* giới hạn tốc độ khung hình 50fps */
#define GAME_FRAME_TIME (1000/GAME_MAX_FPS)

/* enum mô tả dạng va chạm của quả bóng */
typedef enum collide { HIT_NONE, HIT_VERT, HIT_HORZ} COLLIDE;
typedef struct game         /* cấu trúc dữ liệu mô tả trò chơi gồm */
{
    BALL    b;              /* một quả bóng */
    PADDLE  p1, p2;         /* 2 đối tượng paddle đại diện cho 2 người chơi */
    bool    running;        /* cờ báo hiệu trạng thái của trò chơi */
} GAME;

/* khởi tạo trò chơi gồm cửa sổ, renderer, trạng thái ban đầu của trò chơi */
bool gameInit(GAME * g, bool fullscreen);

/* đóng trò chơi */
void gameShutdown(GAME * g);

/* vòng lặp trò chơi */
void gameLoop(GAME * g);

/* cập nhật trạng thái trò chơi */
void gameUpdate(GAME * g);

/* dựng hình cho 1 frame */
void gameDraw(GAME * g);

/* xử lý sự kiện từ bàn phím */
```

```
void gameInput(GAME * g);

/* tính điểm */
void gameGoal(GAME * g, PADDLE * paddle);

/* kiểm tra va chạm giữa các đối tượng trong trò chơi */
COLLIDE gameCheckCollide(GAME * g);
```

- Tập tin GAME.CPP

```
#include "stdafx.h"
#include "game.h"

bool gameInit(GAME * g, bool fullscreen)
{
    time_t t;
    srand((unsigned) time(&t)); /* randomize */
    SDL_Init(SDL_INIT_VIDEO);
    g_window = SDL_CreateWindow("Pong Game",
                                SDL_WINDOWPOS_CENTERED,
                                SDL_WINDOWPOS_CENTERED,
                                BOARD_WIDTH, BOARD_HEIGHT,
                                SDL_WINDOW_SHOWN |
                                (fullscreen ? SDL_WINDOW_FULLSCREEN : 0));
    if(!g_window) return false;

    g_renderer = SDL_CreateRenderer(g_window, -1, SDL_RENDERER_ACCELERATED);
    if (!g_renderer) return false;

    SDL_RenderClear(g_renderer);
    ballInit(&g->b, BALL_START_X, BALL_START_Y);
    paddleInit(&g->p1, PADDLE_LEFT, (BOARD_HEIGHT-PADDLE_HEIGHT)>>1);
    paddleInit(&g->p2, PADDLE_RIGHT, (BOARD_HEIGHT-PADDLE_HEIGHT)>>1);

    g->running = false;
    return true;
}

void gameShutdown(GAME * g)
{
    SDL_DestroyRenderer(g_renderer);
    SDL_DestroyWindow(g_window);
    SDL_Quit();
}

void gameDraw(GAME * g)
{
    /* xóa nội dung framebuffer */
    SDL_SetRenderDrawColor(g_renderer, 0, 0, 0, SDL_ALPHA_OPAQUE);
    SDL_RenderClear(g_renderer);

    /* vẽ màn hình trò chơi */
    rectDraw(0, 0, BOARD_WIDTH, BOARD_CELL, CR_GRAY);
```

```
    rectDraw(0, BOARD_BOTTOM, BOARD_WIDTH, BOARD_CELL, CR_GRAY);
    for (int i = 0; i < (BOARD_HEIGHT/BOARD_CELL); i+=2)
        rectDraw((BOARD_WIDTH-BOARD_CELL)>>1, (i+1)*BOARD_CELL,
                 BOARD_CELL, BOARD_CELL, CR_GRAY);

    /* hiển thị điểm số của 2 người chơi */
    digitDraw((BOARD_WIDTH>>1)-BOARD_CELL*2-DIGIT_WIDTH*BOARD_CELL,
              BOARD_TOP*2, g->p1.score);
    digitDraw((BOARD_WIDTH>>1)+BOARD_CELL*2, BOARD_TOP*2, g->p2.score);

    ballDraw(&g->b);              /* vẽ quả bóng */
    paddleDraw(&g->p1);           /* vẽ ván người chơi 1*/
    paddleDraw(&g->p2);           /* vẽ ván người chơi 2 */
    SDL_RenderPresent(g_renderer); /* copy framebuffer->screen */
}

COLLIDE gameCheckCollide(GAME * g)
{
    /* xây dựng bounding-box cho các đối tượng để kiểm tra vùng chồng lấp */
    SDL_Rect    b  = {g->b.x,  g->b.y,  BALL_SIZE,    BALL_SIZE};
    SDL_Rect    p1 = {g->p1.x, g->p1.y, PADDLE_WIDTH, PADDLE_HEIGHT};
    SDL_Rect    p2 = {g->p2.x, g->p2.y, PADDLE_WIDTH, PADDLE_HEIGHT};

    /* bóng va chạm với 2 đáy */
    if (g->b.y < BOARD_TOP || g->b.y > BALL_COLLIDE_BOTTOM) return HIT_HORZ;

    /* bóng va chạm với mặt trong ván bên trái */
    if (rectCollide(&b, &p1)) {
        g->b.x = g->p1.x+PADDLE_WIDTH;
        return HIT_VERT;
    }
    /* bóng va chạm với mặt trong ván bên phải */
    if (rectCollide(&b, &p2)) {
        g->b.x = g->p2.x-BALL_SIZE;
        return HIT_VERT;
    }
    return HIT_NONE;        /* không phát hiện va chạm */
}

void gameGoal(GAME * g, PADDLE * paddle)
{
    paddle->score++;
    g->b.x = BALL_START_X;  g->b.y = BALL_START_Y;
}

void gameUpdate(GAME * g)
{
    /* cập nhật trạng thái trò chơi */
    paddleUpdate(&g->p1);
    paddleUpdate(&g->p2);
    ballUpdate(&g->b);

    switch(gameCheckCollide(g)) {         /* kiểm tra va chạm */
```

```
    case HIT_HORZ:  g->b.vy *= -1; break;/* va chạm ngang: đảo vy */
    case HIT_VERT:  g->b.vx *= -1; break;/* va chạm dọc: đảo vx */
    default: break;
    }

    /* tính điểm */
    if (g->b.x < 0) gameGoal(g, &g->p2);
    if (g->b.x > BOARD_WIDTH-BALL_SIZE) gameGoal(g, &g->p1);

    /* dừng trò chơi nếu có người chơi đạt đến 10 điểm */
    if (g->p1.score == GAME_MAXSCORE || g->p2.score == GAME_MAXSCORE)
        g->running = false;
}

void gameInput(GAME * g)
{
    SDL_Event e;
    if (SDL_PollEvent(&e)) {
        switch (e.type) {
        case SDL_KEYDOWN:
            switch (e.key.keysym.sym) {
            case SDLK_w: g->p1.dir = PADDLE_UP; break;
            case SDLK_s: g->p1.dir = PADDLE_DOWN; break;
            case SDLK_UP: g->p2.dir = PADDLE_UP; break;
            case SDLK_DOWN: g->p2.dir = PADDLE_DOWN; break;
            case SDLK_ESCAPE: g->running = false; break;
            }
            break;
        case SDL_QUIT:
            g->running = false; break;
        }
    }
}

void gameLoop(GAME * g)
{
    long timerBegin, timerDiff, timerSleep;
    g->running = true;
    while (g->running)
    {
        timerBegin = SDL_GetTicks();
        gameUpdate(g);  /* cập nhật trạng thái trò chơi */
        gameInput(g);   /* xử lý input */
        gameDraw(g);    /* dựng 1 frame hình */
        /* tính toán thời gian dựng 1 frame hình để giới hạn framerate tại 50fps */
        timerDiff = SDL_GetTicks() - timerBegin;
        timerSleep = GAME_FRAME_TIME - timerDiff;
        if(timerSleep > 0)
            SDL_Delay(timerSleep);
    }
}
```

- Tập tin PONG.CPP

```
#include "stdafx.h"
#include "game.h"

#ifdef _MSC_VER         /* thông số của trình liên kết Visual C++ */
#   pragma comment (lib,    "sdl2main.lib")
#   pragma comment (lib,    "sdl2.lib")
#   pragma comment (linker, "/entry:\"mainCRTStartup\"" )
#   pragma comment (linker, "/subsystem:WINDOWS")
#endif

int main(int argc, char ** argv)
{
    GAME game;
    if (!gameInit(&game, false)) return -1;
    gameLoop(&game);
    gameShutdown(&game);
    return 0;
}
```

Mã nguồn

Để tải về mã nguồn của các bài tập này, vui lòng truy cập vào địa chỉ: https://github.com/dzutrinh/CMP211-LAB.