第十二章 项目实战

本章学习目标：

- ✓ 了解 SDL2 库和其相关的辅助库
- ✓ 掌握构建 SDL2 的开发环境
- ✓ 掌握 SDL2 基本绘图方法
- ✓ 掌握 SDL2 事件监控实现方法
- ✓ 掌握 SDL2 简单的游戏编程
- ✓ 掌握一般 C 语言程序的设计思路
- ✓ 掌握程序的调试方法

12.1 实践题

一、SDL2 环境的搭建

实验目的

1．掌握 SDL2 环境搭建

2．测试是否搭建成功

实验步骤

步骤 1：下载 SDL2-devel-2.0.3-VC.zip 开发包，下载地址为 http://www.libsdl.org/release/SDL2-devel-2.0.3-VC.zip,解压下载包。

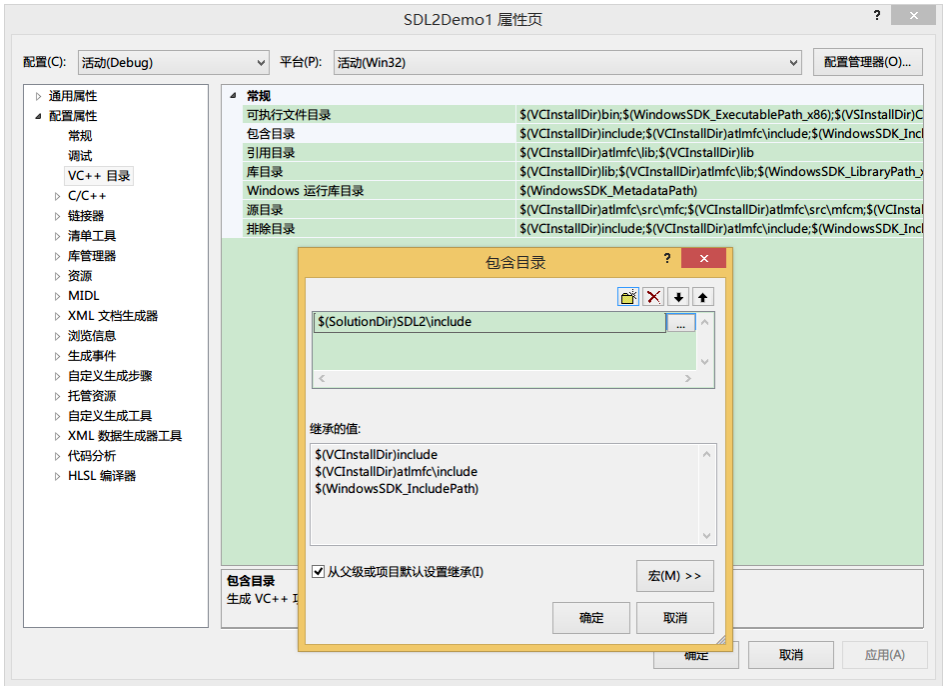
步骤 2：建立解决方案和项目

打开 visual studio 2012,新建 win32 控制台应用程序，解决方案名称为 SDL2Demo1，名称为 SDL2Demo1,点击确定，点击下一步，此处选择"空

项目”选项，如图点击完成。

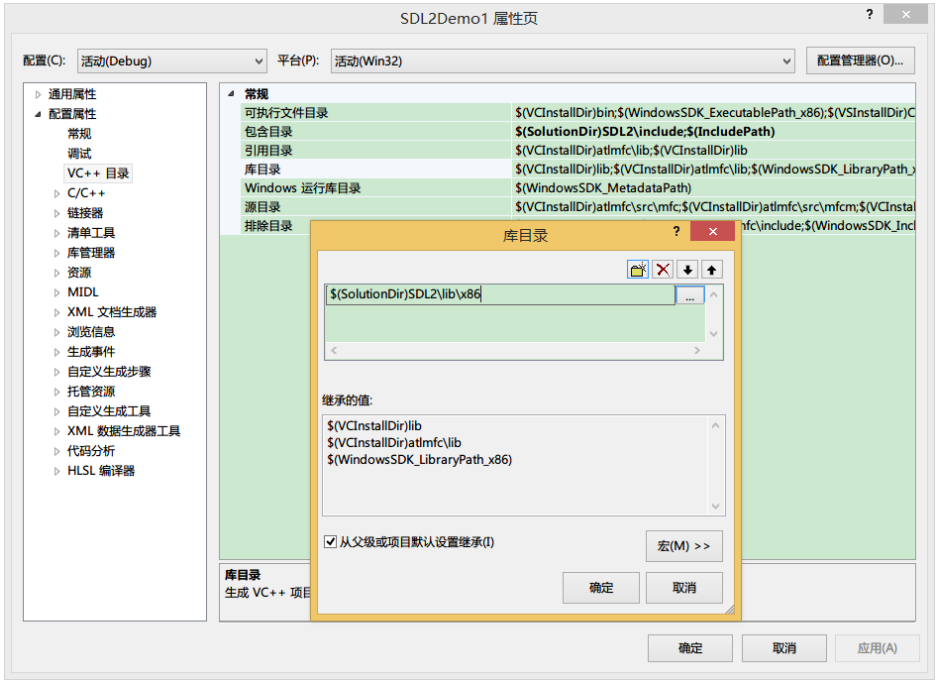
步骤 3：在解决方案路径下新建 SDL2 文件夹，并将 SDL2-devel-2.0.3-VC 解压包下的 include 和 lib 文件夹拷贝到 SDL2 文件夹下。

选择 SDL2Demo1 项目右击选择属性，导航到 VC++目录，选择右侧的包含目录并新建一项，内容为$(SolutionDir)SDL2\include，点击确定。

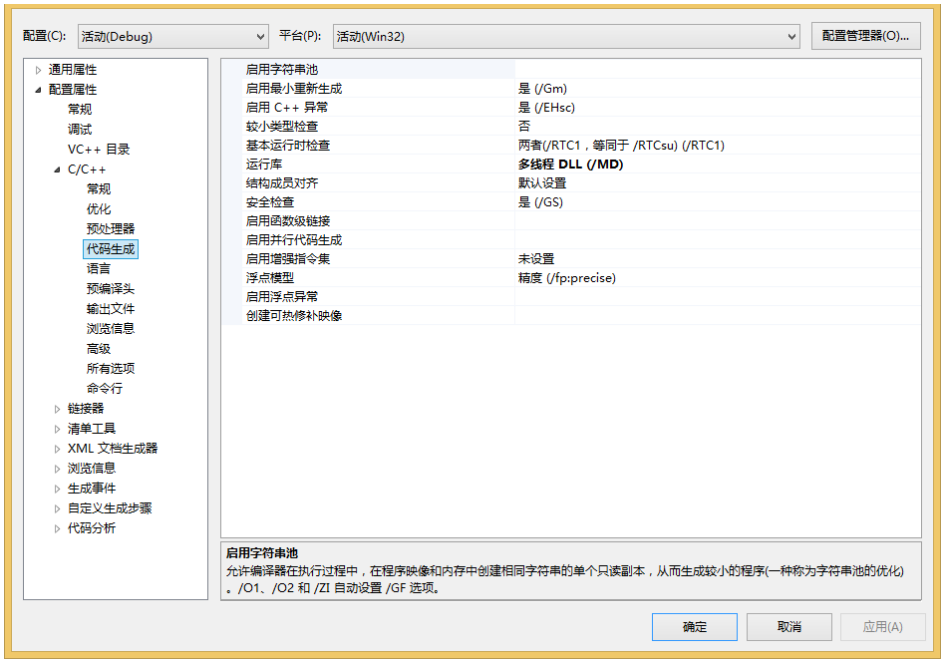


选择右侧的库目录并新建一项，内容为

$(SolutionDir)SDL2\lib\x86，点击确定。

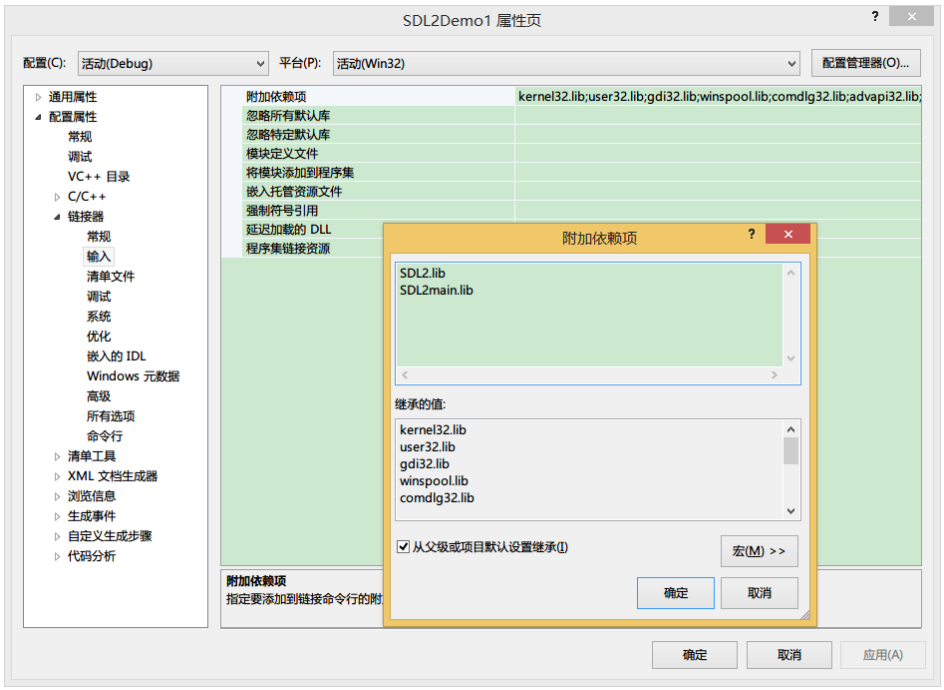


导航到 C/C++下的代码生成，右侧选择运行库改选项为”多线程 DLL (/MD)”，点击应用。

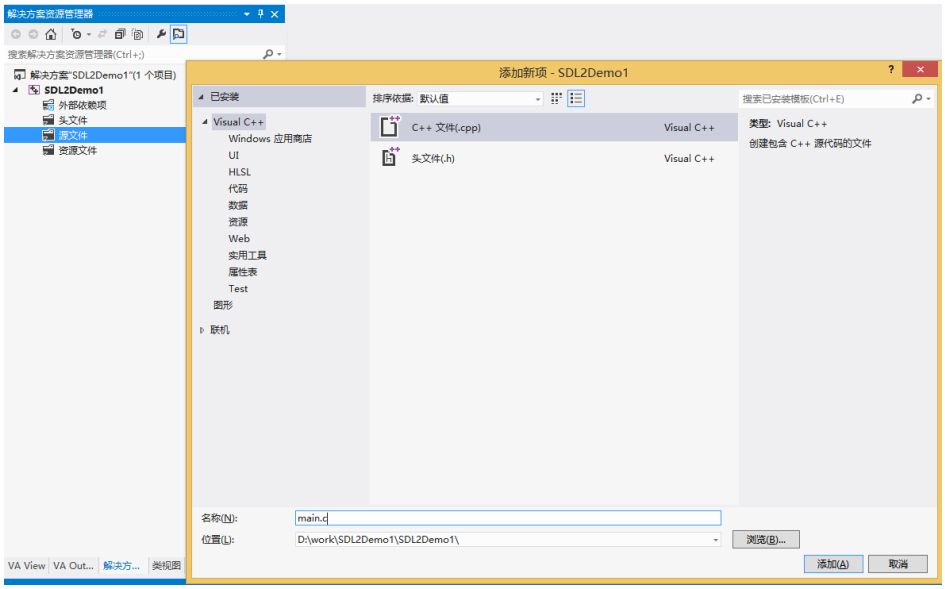


导航到链接器下的输入选项，右侧选择附加依赖项，添加内容为

SDL2.lib、SDL2main.lib，点击确定。



步骤 4：在 SDL2Demo1 项目中选择源文件，右击添加新建项，导航到 Visual C++ 下的 C++文件(.cpp)，名称为 main.c，点击添加。

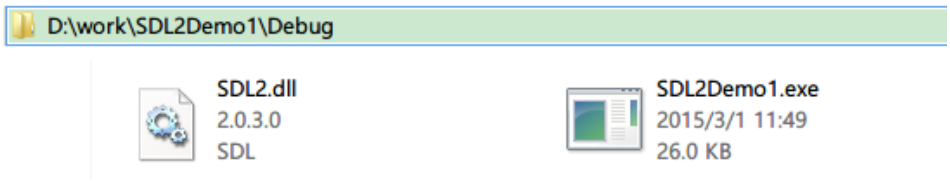


该文件内容为：

```
int main(int argc,char** argv)
{
    return 0;
}
```

选择项目生成一下该项目，拷贝解压包 SDL2-2.0.3\lib\x86 路径下的

SDL2.dll 到解决方案的 Debug 目录中。

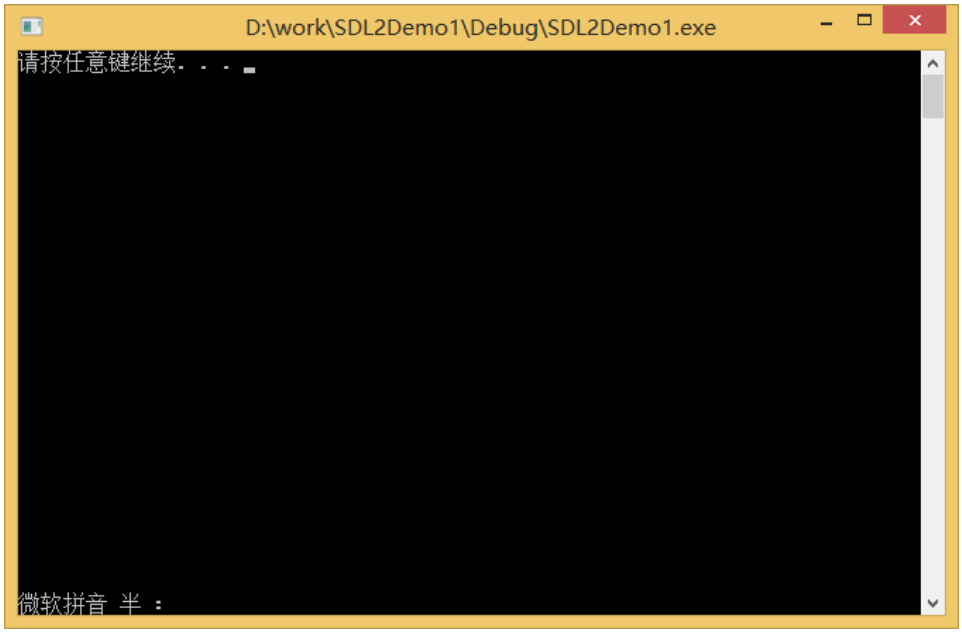


步骤 5：编写测试代码，修改 main.c 文件中的代码

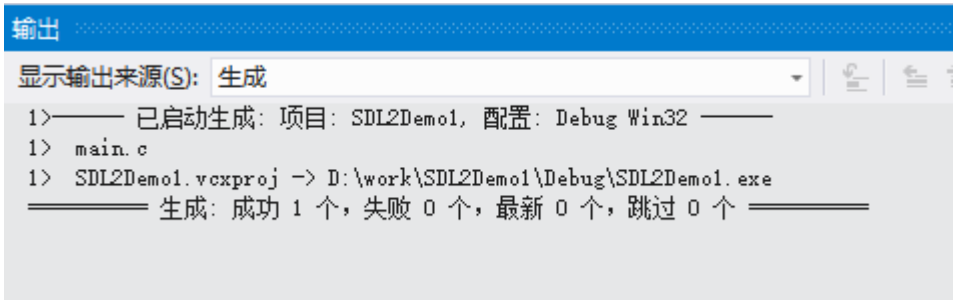
```
#include <stdlib.h>
#include <stdio.h>
#include "SDL.h"

int main(int argc,char** argv)
{
    //初始化 SDL 各个子系统
    if((SDL_Init(SDL_INIT_EVERYTHING)==-1)) {
        //初始化失败，则程序直接退出
        exit(-1);
}
    //关闭 SDL 各个子系统
    SDL_Quit();
    system("pause");
    return 0;
}
```

编译运行程序，如果程序直接退出表示初始化失败，如果显现如下界面表示

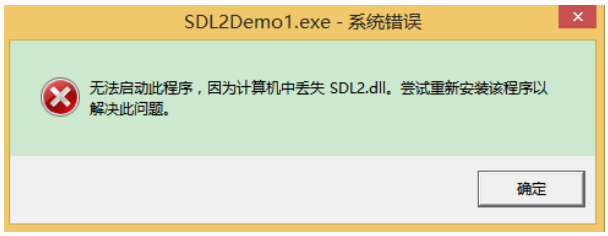
程序运行正常。



步骤 6：删除解决方案目录下 Debug 文件夹中的 SDL2.dll 文件，编译项目，发现编译成功。



运行失败。



步骤 7：选择 SDL2Demo1 项目的 release 版本进行编译运行，发现编译失败。

实验结果/结论

1. 实验结果

✓ 实验步骤 6 表明程序的执行需要 SDL2.dll 文件。

✓ 实验步骤 7 如果要编译成功需要对 release 编译环境进行设置。

2. 实验结论

- ✓ SDL2 库使用的方法是导入库(lib 文件)和动态库(dll 文件)配合，导入库只在编译和链接阶段起作用，动态库在运行阶段起作用。
- ✓ Debug 版本和 Release 版本都需要分别进行设置才可以编译运行。

二、SDL2 显示 bmp 图片

实验目的

1．掌握通过 SDL2 库显示一副 bmp 图片的流程

2．掌握通过 SDL2 显示图片的一些函数

实验步骤

步骤 1：按照试验一所示步骤搭建开发环境，建立的项目名称为 SDL2App

步骤 2：在解决方案目录下新建名称为 res 的文件夹，将准备的好的图片 start.bmp 和 back.bmp 拷贝到该目录下。

步骤 3：打开 main.c 源文件，修改代码内容为：

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <memory.h>
#include "SDL.h"

#define G_WINDOW_X 50
#define G_WINDOW_Y 50
#define G_WINDOW_W 800
#define G_WINDOW_H 600

typedef struct LoadedPicture {
    SDL_Surface *surface;
    SDL_Texture *texture;
    const char* name;
} LoadedPicture;

int main(int argc, char** argv)
```

```
{
    Uint8 num_pictures;
    LoadedPicture* pictures;
    SDL_PixelFormat* format = NULL;
    SDL_Window *window;
    SDL_Renderer *renderer;
    Uint32 pixelFormat = 0;
    int access = 0;
    SDL_Rect srcrect;
    SDL_Rect dstrect;
    int i;
    int j;
    char *game[] = {
        "res\\back.bmp",
        "res\\start.bmp"
    };
    SDL_LogSetPriority(SDL_LOG_CATEGORY_APPLICATION,
    SDL_LOG_PRIORITY_INFO);
    if (SDL_Init(SDL_INIT_EVERYTHING) == -1) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could
    not initialize SDL.");
        exit(-2);
    }
    num_pictures = 2;
    pictures = (LoadedPicture
    *)SDL_malloc(sizeof(LoadedPicture)*num_pictures);
    for (i = 0; i < num_pictures; i++)
        pictures[i].surface = NULL;
    for (i = 0; i < num_pictures; i++) {
```

```
        pictures[i].surface = SDL_LoadBMP(game[i]);
        pictures[i].name = game[i];
        if (pictures[i].surface == NULL) {
            j = 0;
            for (j = 0; j < num_pictures; j++)
                SDL_FreeSurface(pictures[j].surface);
            SDL_free(pictures);
            SDL_Quit();
            SDL_LogError(SDL_LOG_CATEGORY_APPLICATION,
    "Could not load surface from named bitmap file: %s",
    game[i]);
            exit(-3);
        }
    }
    window = SDL_CreateWindow("Demo", G_WINDOW_X,
    G_WINDOW_Y, G_WINDOW_W, G_WINDOW_H, 0);
    SDL_SetWindowPosition(window, G_WINDOW_X, G_WINDOW_Y);
    if (window == NULL) {
        for (i = 0; i < num_pictures; i++)
            SDL_FreeSurface(pictures[i].surface);
        SDL_free(pictures);
        SDL_Quit();
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could
    not create window for Demo.");
        exit(-4);
    }
    renderer = SDL_CreateRenderer(window, -1, 0);
    if (!renderer) {
        SDL_DestroyWindow(window);
```

```
            for (i = 0; i < num_pictures; i++)
                SDL_FreeSurface(pictures[i].surface);
            SDL_free(pictures);
            SDL_Quit();
            SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could
        not create rendering context for Demo window.");
            exit(-5);
        }
        for (i = 0; i < num_pictures; i++)
            pictures[i].texture = NULL;
        for (i = 0; i < num_pictures; i++) {
            pictures[i].texture =
        SDL_CreateTextureFromSurface(renderer,
        pictures[i].surface);
            if (pictures[i].texture == NULL) {
                j = 0;
                for (j = 0; j < num_pictures; i++)
                    if (pictures[i].texture != NULL)
                        SDL_DestroyTexture(pictures[i].texture);
                for (i = 0; i < num_pictures; i++)
                    SDL_FreeSurface(pictures[i].surface);
                SDL_free(pictures);
                SDL_DestroyRenderer(renderer);
                SDL_DestroyWindow(window);
                SDL_Quit();
                SDL_LogError(SDL_LOG_CATEGORY_APPLICATION,
        "Could not create texture for Demo.");
                exit(-6);
            }
```

```
    }

    SDL_RenderCopy(renderer, pictures[0].texture, NULL,
    NULL);
    SDL_RenderPresent(renderer);

    srcrect.x = 10;
    srcrect.y = 10;
    srcrect.h = G_WINDOW_H / 2;
    srcrect.w = G_WINDOW_W / 2;

    dstrect.x = 10;
    dstrect.y = 10;
    dstrect.h = G_WINDOW_H / 2;
    dstrect.w = G_WINDOW_W / 2;
    SDL_RenderCopy(renderer, pictures[1].texture, &srcrect,
    &dstrect);
    SDL_RenderPresent(renderer);
    SDL_Delay(2000);

    srcrect.x = 0;
    srcrect.y = 0;
    srcrect.h = G_WINDOW_H;
    srcrect.w = G_WINDOW_W;

    dstrect.x = 0;
    dstrect.y = 0;
    dstrect.h = G_WINDOW_H;
    dstrect.w = G_WINDOW_W;
```

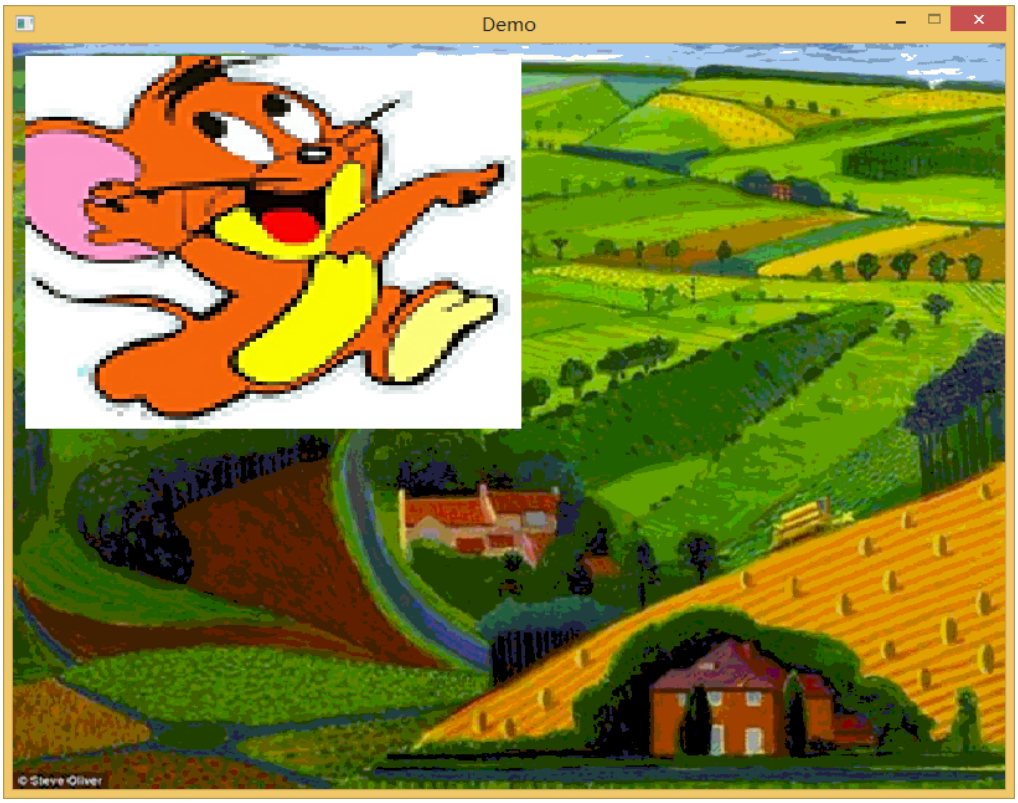
```
SDL_RenderClear(renderer);
SDL_RenderCopy(renderer, pictures[0].texture, &srcrect,
&dstrect);
SDL_RenderPresent(renderer);

srcrect.x = 10;
srcrect.y = 10;
srcrect.h = G_WINDOW_H / 2;
srcrect.w = G_WINDOW_W / 2;

dstrect.x = 10;
dstrect.y = 10;
dstrect.h = G_WINDOW_H / 2;
dstrect.w = G_WINDOW_W / 2;
SDL_RenderCopy(renderer, pictures[1].texture, &srcrect,
&dstrect);
SDL_RenderPresent(renderer);
SDL_Delay(2000);

system("pause");
for (i = 0; i < num_pictures; i++)
    SDL_DestroyTexture(pictures[i].texture);
SDL_DestroyRenderer(renderer);
SDL_DestroyWindow(window);
for (i = 0; i < num_pictures; i++)
    SDL_FreeSurface(pictures[i].surface);
SDL_free(pictures);
SDL_Quit();
```

```
        return 0;
    }
```

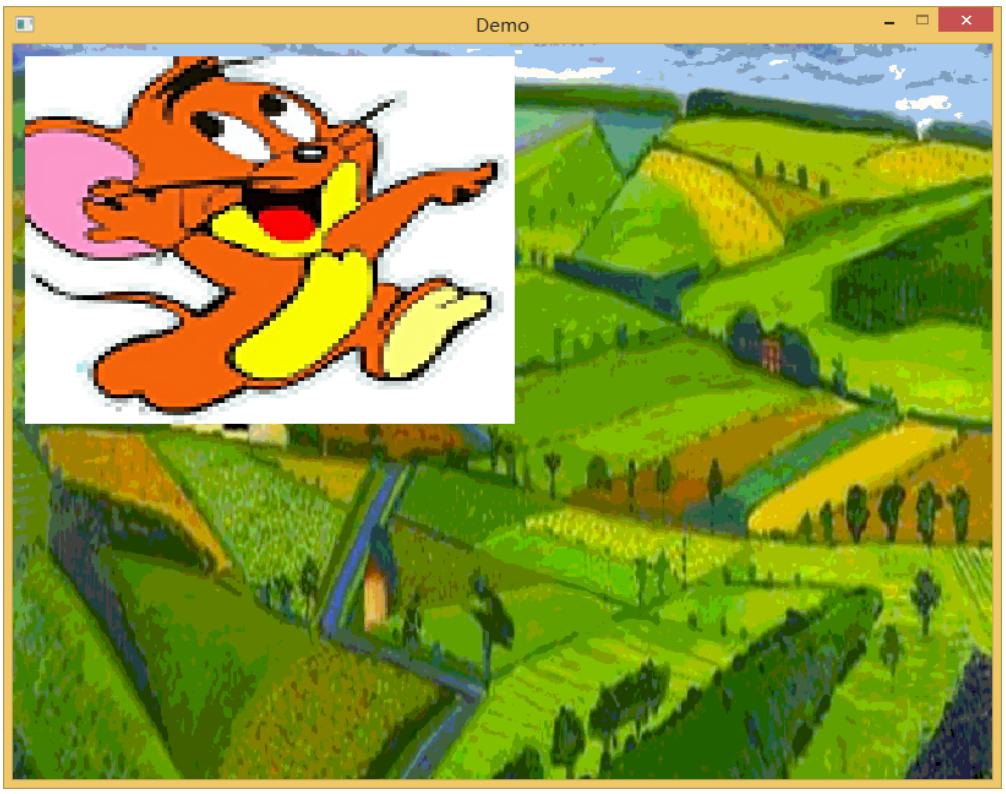
实验结果/结论

1. 实验结果

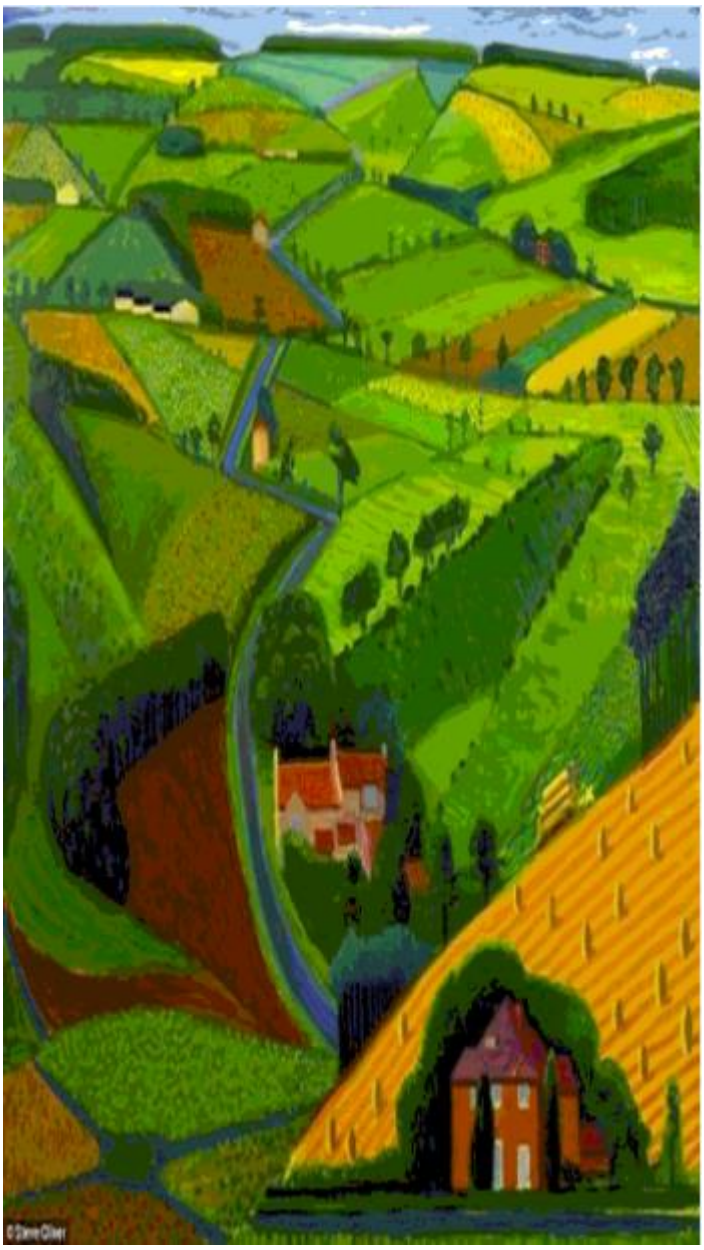
✓ 程序运行后显示界面为



延迟 2 秒后显示



back.bmp 原始图片为



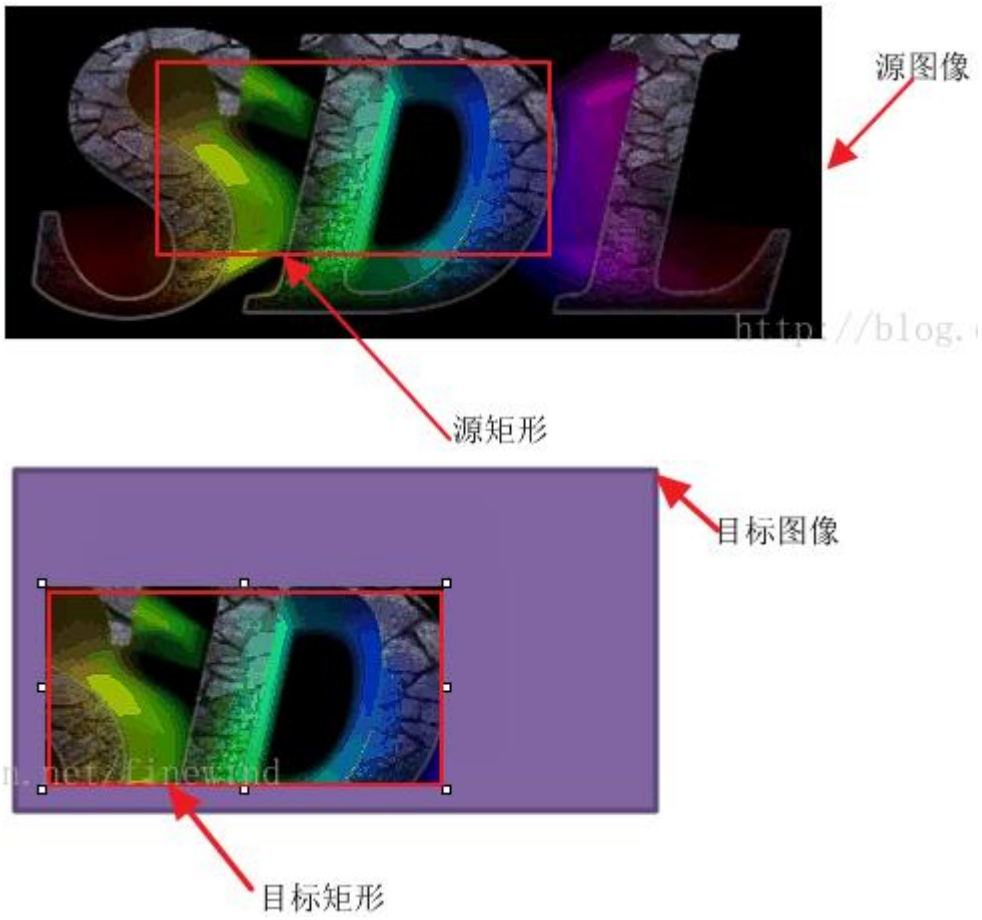
start.bmp 的原始图片为

2． 实验结论

✓ 通过 SDL2 显示图片的主要流程为首先加载图片数据到 SDL_Surface 然后通过 SDL_CreateTextureFromSurface 创建 SDL_Texture，通过 SDL_RenderCopy 和 SDL_RenderPresent 显示图片，SDL 使用 SDL_Surface 和 SDL_Texture 这 2 种结构绘图到屏幕。

✓ SDL_RenderCopy 函数，源矩形用于指定要拷贝到目标图像的源图像起始位置及宽高参数，即可以只拷贝源图像的某一部分到目标图像。

目标矩形用于指定要拷贝到目标图像中的起始位置及宽高参数，即可以通过指定不同的目标矩形位置，让源图像拷贝到目标图像的不同位置。且目标矩形的宽高不用和源矩形一致，这可用于缩小或放大图像。x,y 为矩形的开始位置；w,h 为矩形的宽高。



✓ 程序启动后会显示 SDL 的日志窗口，便于程序开发和调试。

三、SDL2 监控键盘和鼠标事件

实验目的

1．掌握 SDL2 如何监控鼠标的点击事件和监控鼠标的坐标

2．掌握 SDL2 如何监控键盘按键事件

实验步骤

步骤 1：按照试验一所示步骤搭建开发环境，建立的项目名称为 SDL2CtrlApp

步骤 2：打开 main.c 源文件，修改代码内容为：

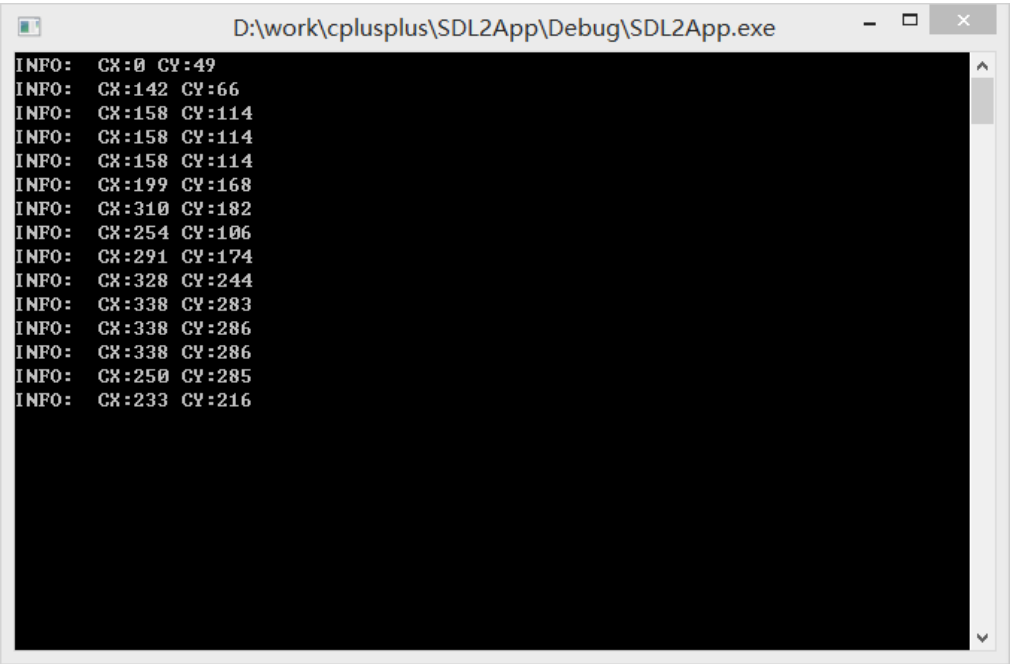
```c
#include <stdlib.h>
#include <stdio.h>
#include "SDL.h"

#define G_WINDOW_X 50
#define G_WINDOW_Y 50
#define G_WINDOW_W 750
#define G_WINDOW_H 550

int main(int argc, char** argv)
{
    SDL_PixelFormat* format = NULL;
    SDL_Window *window;
    SDL_Event event;
    int event_pending = 0;
    int should_exit = 0;
    int button_down;
    int cx;
    int cy;
    SDL_LogSetPriority(SDL_LOG_CATEGORY_APPLICATION,
    SDL_LOG_PRIORITY_INFO);
    if (SDL_Init(SDL_INIT_EVERYTHING) == -1) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could
    not initialize SDL");
        exit(-2);
    }
    window = SDL_CreateWindow("Event", G_WINDOW_X,
    G_WINDOW_Y, G_WINDOW_W, G_WINDOW_H, 0);
    SDL_SetWindowPosition(window, G_WINDOW_X, G_WINDOW_Y);
```

```
        event_pending = 0;
        should_exit = 0;
        event_pending = SDL_PollEvent(&event);
        while (should_exit == 0) {
            event_pending = SDL_PollEvent(&event);
            if (event_pending == 1) {
                if (event.type == SDL_KEYDOWN){
                    button_down = 1;
                    if (event.key.keysym.sym == SDLK_ESCAPE) {
                        should_exit = 1;
                        break;
                    }
                }
                if (event.type == SDL_MOUSEBUTTONDOWN){
                    cx = event.button.x;
                    cy = event.button.y;
                    SDL_LogInfo(SDL_LOG_CATEGORY_APPLICATION, "
        CX:%d CY:%d ",cx,  cy);
                }
                if (event.type == SDL_QUIT)
                    should_exit = 1;
                event_pending = 0;
            }
        }
        SDL_DestroyWindow(window);
        SDL_Quit();
        return 0;
    }
```

实验结果/结论

1. 实验结果

✓ 运行程序会启动程序的主窗口和日志窗口，在主窗口中单击鼠标会在日志窗口中提示鼠标的坐标值，这个坐标值是相对于主窗口的。



坐标通过 SDL_MouseButtonEvent 结构体变量 button 获得。

✓ 将窗体控制焦点给主窗口，点击键盘的 ESC 键，整个程序将退出。我们通过查看 SDLK_ESCAPE 的宏定义可以发现 SDL_keycode.h 文件，该文件中有很多键的定义。我们程序中使用 event.key.keysym.sym == SDLK_ESCAPE 进行按键的判断。

2. 实验结论

✓ SDL2 中事件有专门的类型 SDL_Event 类型，SDL_Event 是一个联合体，其定义如下：

```
typedef union SDL_Event
{
    Uint32 type;                    /**< Event type, shared with
all events */
    SDL_CommonEvent common;         /**< Common event data */
    SDL_WindowEvent window;         /**< Window event data */
    SDL_KeyboardEvent key;          /**< Keyboard event data */
    SDL_TextEditingEvent edit;      /**< Text editing event data
*/
    SDL_TextInputEvent text;        /**< Text input event data
*/
    SDL_MouseMotionEvent motion;    /**< Mouse motion event data
*/
```

```
    SDL_MouseButtonEvent button;    /**< Mouse button event data
*/
    SDL_MouseWheelEvent wheel;      /**< Mouse wheel event data
*/
    SDL_JoyAxisEvent jaxis;         /**< Joystick axis event
data */
    SDL_JoyBallEvent jball;         /**< Joystick ball event
data */
    SDL_JoyHatEvent jhat;           /**< Joystick hat event data
*/
    SDL_JoyButtonEvent jbutton;     /**< Joystick button event
data */
    SDL_JoyDeviceEvent jdevice;     /**< Joystick device change
event data */
    SDL_ControllerAxisEvent caxis;      /**< Game Controller
axis event data */
    SDL_ControllerButtonEvent cbutton;  /**< Game Controller
button event data */
    SDL_ControllerDeviceEvent cdevice;  /**< Game Controller
device event data */
    SDL_QuitEvent quit;             /**< Quit request event data
*/
    SDL_UserEvent user;             /**< Custom event data */
    SDL_SysWMEvent syswm;           /**< System dependent window
event data */
    SDL_TouchFingerEvent tfinger;   /**< Touch finger event data
*/
    SDL_MultiGestureEvent mgesture; /**< Gesture event data */
    SDL_DollarGestureEvent dgesture; /**< Gesture event data */
    SDL_DropEvent drop;             /**< Drag and drop event
data */

    /* This is necessary for ABI compatibility between Visual
C++ and GCC
       Visual C++ will respect the push pack pragma and use 52
bytes for
       this structure, and GCC will use the alignment of the
largest datatype
       within the union, which is 8 bytes.

       So... we'll add padding to force the size to be 56 bytes
for both.
    */
    Uint8 padding[56];
```

```
    } SDL_Event;
```

其中 type 字段决定了是那种事件，是一个枚举类型，其定义为：

```
typedef enum
{
    SDL_FIRSTEVENT     = 0,     /**< Unused (do not remove) */

    /* Application events */
    SDL_QUIT           = 0x100, /**< User-requested quit */

    /* These application events have special meaning on iOS, see
README-ios.txt for details */
    SDL_APP_TERMINATING,        /**< The application is being
terminated by the OS
                                     Called on iOS in
applicationWillTerminate()
                                     Called on Android in
onDestroy()
                                */
    SDL_APP_LOWMEMORY,          /**< The application is low on
memory, free memory if possible.
                                     Called on iOS in
applicationDidReceiveMemoryWarning()
                                     Called on Android in
onLowMemory()
                                */
    SDL_APP_WILLENTERBACKGROUND, /**< The application is about to
enter the background
                                     Called on iOS in
applicationWillResignActive()
                                     Called on Android in onPause()
                                */
    SDL_APP_DIDENTERBACKGROUND, /**< The application did enter the
background and may not get CPU for some time
                                     Called on iOS in
applicationDidEnterBackground()
                                     Called on Android in onPause()
                                */
    SDL_APP_WILLENTERFOREGROUND, /**< The application is about to
enter the foreground
                                     Called on iOS in
applicationWillEnterForeground()
                                     Called on Android in onResume()
                                */
    SDL_APP_DIDENTERFOREGROUND, /**< The application is now
```

```
interactive
                                    Called on iOS in
applicationDidBecomeActive()
                                    Called on Android in onResume()
                                */

    /* Window events */
    SDL_WINDOWEVENT    = 0x200, /**< Window state change */
    SDL_SYSWMEVENT,             /**< System specific event */

    /* Keyboard events */
    SDL_KEYDOWN        = 0x300, /**< Key pressed */
    SDL_KEYUP,                  /**< Key released */
    SDL_TEXTEDITING,            /**< Keyboard text editing
(composition) */
    SDL_TEXTINPUT,              /**< Keyboard text input */

    /* Mouse events */
    SDL_MOUSEMOTION    = 0x400, /**< Mouse moved */
    SDL_MOUSEBUTTONDOWN,        /**< Mouse button pressed */
    SDL_MOUSEBUTTONUP,          /**< Mouse button released */
    SDL_MOUSEWHEEL,             /**< Mouse wheel motion */

    /* Joystick events */
    SDL_JOYAXISMOTION  = 0x600, /**< Joystick axis motion */
    SDL_JOYBALLMOTION,          /**< Joystick trackball motion */
    SDL_JOYHATMOTION,           /**< Joystick hat position change */
    SDL_JOYBUTTONDOWN,          /**< Joystick button pressed */
    SDL_JOYBUTTONUP,            /**< Joystick button released */
    SDL_JOYDEVICEADDED,         /**< A new joystick has been
inserted into the system */
    SDL_JOYDEVICEREMOVED,       /**< An opened joystick has been
removed */

    /* Game controller events */
    SDL_CONTROLLERAXISMOTION  = 0x650, /**< Game controller axis
motion */
    SDL_CONTROLLERBUTTONDOWN,          /**< Game controller button
pressed */
    SDL_CONTROLLERBUTTONUP,            /**< Game controller button
released */
    SDL_CONTROLLERDEVICEADDED,         /**< A new Game controller
has been inserted into the system */
    SDL_CONTROLLERDEVICEREMOVED,       /**< An opened Game
```

```
controller has been removed */
    SDL_CONTROLLERDEVICEREMAPPED,       /**< The controller mapping
was updated */

    /* Touch events */
    SDL_FINGERDOWN      = 0x700,
    SDL_FINGERUP,
    SDL_FINGERMOTION,

    /* Gesture events */
    SDL_DOLLARGESTURE   = 0x800,
    SDL_DOLLARRECORD,
    SDL_MULTIGESTURE,

    /* Clipboard events */
    SDL_CLIPBOARDUPDATE = 0x900, /**< The clipboard changed */

    /* Drag and drop events */
    SDL_DROPFILE        = 0x1000, /**< The system requests a file
open */

    /* Render events */
    SDL_RENDER_TARGETS_RESET = 0x2000, /**< The render targets have
been reset */

    /** Events ::SDL_USEREVENT through ::SDL_LASTEVENT are for your
use,
     *  and should be allocated with SDL_RegisterEvents()
     */
    SDL_USEREVENT    = 0x8000,

    /**
     *  This last event is only for bounding internal arrays
     */
    SDL_LASTEVENT    = 0xFFFF
} SDL_EventType;
```

SDL_KEYDOWN 表示按下某键， SDL_KEYUP，表示松开某键，SDL_MOUSEMOTION,表示鼠标移动，SDL_MOUSEBUTTONDOWN，表示鼠标键按下，SDL_MOUSEBUTTONUP，表示鼠标键松开。

✓ event.button 保存的是鼠标相关的值，该结构体定义为

```
    typedef struct SDL_MouseButtonEvent
    {
        Uint32 type;        /**< ::SDL_MOUSEBUTTONDOWN
```

```
or ::SDL_MOUSEBUTTONUP */
    Uint32 timestamp;
    Uint32 windowID;    /**< The window with mouse focus, if any
*/
    Uint32 which;       /**< The mouse instance id, or
SDL_TOUCH_MOUSEID */
    Uint8 button;       /**< The mouse button index */
    Uint8 state;        /**< ::SDL_PRESSED or ::SDL_RELEASED */
    Uint8 clicks;       /**< 1 for single-click, 2 for double-
click, etc. */
    Uint8 padding1;
    Sint32 x;           /**< X coordinate, relative to window */
    Sint32 y;           /**< Y coordinate, relative to window */
} SDL_MouseButtonEvent;
```

其中 x，y 分别为鼠标的坐标值。

四、SDL2 实现打地鼠游戏

实验目的

1．掌握 SDL2 图像库扩展库的使用

2．掌握使用 SDL2 进行小型游戏的开发

实验步骤

步骤 1：按照试验一所示步骤搭建开发环境，建立的项目名称为 WhacAMole

步骤 2：将预先准备好的 png 图片素材拷贝到项目的路径下，下载 SDL2 的 image 库 http://www.libsdl.org/projects/SDL_image/release/SDL2_image-devel-2.0.0-VC.zip 并按照试验一整合到开发环境之中。

步骤 3：程序的逻辑为随机的在田野中出现 3 只鼠，使用鼠标点击鼠，如果检测到锤子和鼠有碰撞则显示出锤子，添加 main.c 编写如下代码

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <memory.h>
#include <time.h>
#include "SDL.h"
#include "SDL_image.h"

#define G_WINDOW_X 10
#define G_WINDOW_Y 10
#define G_WINDOW_W 600
#define G_WINDOW_H 500
```

```
#define G_MOUSE_W 90
#define G_MOUSE_H 120
#define G_CUT_W 90
#define G_CUT_H 120

typedef struct LoadedPicture {
    SDL_Surface *surface;
    SDL_Texture *texture;
    const char* name;
} LoadedPicture;

void render(SDL_Renderer *renderer, SDL_Texture *texture,
SDL_Rect texture_dimensions)
{
    SDL_RenderCopy(renderer, texture, &texture_dimensions,
&texture_dimensions);
    SDL_RenderPresent(renderer);
}

void renderTexture(SDL_Renderer *renderer, SDL_Texture *texture,
int x, int y) {
    SDL_Rect dst;
    dst.x = x;
    dst.y = y;
    SDL_QueryTexture(texture, NULL, NULL, &dst.w, &dst.h);
    SDL_RenderCopy(renderer, texture, &dst, &dst);
    SDL_RenderPresent(renderer);
}

int main(int argc, char** argv)
{
    Uint8 num_pictures;
    LoadedPicture* pictures;
    SDL_PixelFormat* format = NULL;
    SDL_Window *window;
    SDL_Renderer *renderer;
    SDL_Color black = { 0, 0, 0, 0xff };
    SDL_Event event;
    int event_pending = 0;
    int should_exit = 0;
    unsigned int current_picture;
    int button_down;
    Uint32 pixelFormat = 0;
    int access = 0;
```

```
    SDL_Rect texture_dimensions;
    int mx;
    int my;
    int cx;
    int cy;
    int kn;
    int mxx;
    int myy;
    int i;
    int j;
    int mxy[3][2];
    time_t start_time;
    time_t end_time;
    double elapsed_time;
    char *game[] = {
        "back.png",
        "mouse.png",
        "cut.png",
        "start.png",
        "jerry.png"
    };
    mx = 0;
    my = 0;
    cx = 0;
    cy = 0;
    kn = 0;
    srand((unsigned)time(NULL));
    SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
SDL_GL_CONTEXT_PROFILE_ES);
    SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 2);
    SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 0);
    SDL_LogSetPriority(SDL_LOG_CATEGORY_APPLICATION,
SDL_LOG_PRIORITY_INFO);
    if (SDL_Init(SDL_INIT_EVERYTHING) == -1) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could not
initialize SDL.");
        exit(-2);
    }
    if (IMG_Init(IMG_INIT_PNG) == -1)
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could not
initialize SDL IMG.");
        exit(-2);
    }
```

```
    num_pictures = 5;
    pictures = (LoadedPicture
*)SDL_malloc(sizeof(LoadedPicture)*num_pictures);
    for (i = 0; i < num_pictures; i++)
        pictures[i].surface = NULL;
    for (i = 0; i < num_pictures; i++) {
        //pictures[i].surface = SDL_LoadBMP(game[i]);
        pictures[i].surface = IMG_Load(game[i]);
        pictures[i].name = game[i];
        if (pictures[i].surface == NULL) {
            j = 0;
            for (j = 0; j < num_pictures; j++)
                SDL_FreeSurface(pictures[j].surface);
            SDL_free(pictures);
            SDL_Quit();
            SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could
not load surface from named file: %s", game[i]);
            exit(-3);
        }
    }
    window = SDL_CreateWindow("Demo", G_WINDOW_X, G_WINDOW_Y,
G_WINDOW_W, G_WINDOW_H, 0);
    //window = SDL_CreateWindow("Whac-A-
Mole",SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED,G_WINDOW_W,G_WINDOW_H,SDL_WINDOW_FULLSCR
EEN);
    //window = SDL_CreateWindow("Whac-A-Mole",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOW_FULLSCREEN);
    if (window == NULL) {
        for (i = 0; i < num_pictures; i++)
            SDL_FreeSurface(pictures[i].surface);
        SDL_free(pictures);
        SDL_Quit();
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could not
create window for SDL_GAME.");
        exit(-4);
    }
    renderer = SDL_CreateRenderer(window, -1, 0);
    if (!renderer) {
        SDL_DestroyWindow(window);
        for (i = 0; i < num_pictures; i++)
            SDL_FreeSurface(pictures[i].surface);
```

```
        SDL_free(pictures);
        SDL_Quit();
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could not
create rendering context for SDL_GAME window.");
        exit(-5);
    }
    for (i = 0; i < num_pictures; i++)
        pictures[i].texture = NULL;
    for (i = 0; i < num_pictures; i++) {
        pictures[i].texture =
SDL_CreateTextureFromSurface(renderer, pictures[i].surface);
        if (pictures[i].texture == NULL) {
            j = 0;
            for (j = 0; j < num_pictures; i++)
                if (pictures[i].texture != NULL)
                    SDL_DestroyTexture(pictures[i].texture);
            for (i = 0; i < num_pictures; i++)
                SDL_FreeSurface(pictures[i].surface);
            SDL_free(pictures);
            SDL_DestroyRenderer(renderer);
            SDL_DestroyWindow(window);
            SDL_Quit();
            SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could
not create texture for SDL_shape.");
            exit(-6);
        }
    }
    event_pending = 0;
    should_exit = 0;
    event_pending = SDL_PollEvent(&event);
    current_picture = 0;
    button_down = 0;
    texture_dimensions.h = 0;
    texture_dimensions.w = 0;
    texture_dimensions.x = 0;
    texture_dimensions.y = 0;
    SDL_QueryTexture(pictures[3].texture, (Uint32
*)&pixelFormat, (int *)&access, &texture_dimensions.w,
&texture_dimensions.h);
    render(renderer, pictures[3].texture, texture_dimensions);
    SDL_RenderClear(renderer);
    SDL_RenderCopy(renderer, pictures[3].texture,
&texture_dimensions, &texture_dimensions);
    SDL_Delay(2 * 1000);
```

```
    SDL_QueryTexture(pictures[current_picture].texture, (Uint32
*)&pixelFormat, (int *)&access, &texture_dimensions.w,
&texture_dimensions.h);
    SDL_RenderClear(renderer);
    SDL_RenderCopy(renderer, pictures[current_picture].texture,
&texture_dimensions, &texture_dimensions);
    time(&start_time);
    while (should_exit == 0) {
        time(&end_time);
        elapsed_time = difftime(end_time, start_time);
        if (elapsed_time > 2)
        {
            time(&start_time);
            texture_dimensions.h = G_WINDOW_H;
            texture_dimensions.w = G_WINDOW_W;
            texture_dimensions.x = 0;
            texture_dimensions.y = 0;
            SDL_RenderClear(renderer);
            SDL_QueryTexture(pictures[0].texture, (Uint32
*)&pixelFormat, (int *)&access, &texture_dimensions.w,
&texture_dimensions.h);
            texture_dimensions.x = 0;
            texture_dimensions.y = 0;
            SDL_RenderCopy(renderer, pictures[0].texture,
&texture_dimensions, &texture_dimensions);
            for (i = 0; i < 3; i++)
            {
                mx = rand() % (G_WINDOW_W - G_MOUSE_W);
                my = rand() % (G_WINDOW_H - G_MOUSE_H);
                texture_dimensions.h = G_MOUSE_H;
                texture_dimensions.w = G_MOUSE_W;
                texture_dimensions.x = mx;
                texture_dimensions.y = my;
                if (i == 1)
                {
                    SDL_RenderCopy(renderer,
pictures[4].texture, NULL, &texture_dimensions);
                }
                else{
                    SDL_RenderCopy(renderer,
pictures[1].texture, NULL, &texture_dimensions);
                }
                mxy[i][0] = mx;
                mxy[i][1] = my;
```

```
            }
            SDL_RenderPresent(renderer);
        }
        event_pending = SDL_PollEvent(&event);
        if (event_pending == 1) {
            if (event.type == SDL_KEYDOWN){
                button_down = 1;
                if (event.key.keysym.sym == SDLK_ESCAPE ||
event.key.keysym.sym == SDLK_KP_BACKSPACE) {
                    should_exit = 1;
                    break;
                }
            }
            if (button_down && event.type == SDL_KEYUP) {
                SDL_LogInfo(SDL_LOG_CATEGORY_APPLICATION,
"Changing to shaped bmp: %s", pictures[current_picture].name);
            }
            if (event.type == SDL_MOUSEBUTTONDOWN){
                cx = event.button.x;
                cy = event.button.y;
                for (i = 0; i < 3; i++)
                {
                    mx = mxy[i][0];
                    my = mxy[i][1];
                    mxx = mx + G_MOUSE_W / 2;
                    myy = my + G_MOUSE_H / 2;
                    SDL_LogInfo(SDL_LOG_CATEGORY_APPLICATION,
"MX: %d  CX:%d  MY: %d CY:%d  abs(X):%d abs(Y):%d", mx, cx, my,
cy, abs(mx - cx), abs(my - cy));
                    if ((abs(cx - mxx) < (G_MOUSE_W + G_CUT_W) /
2) && (abs(cy - myy) < (G_MOUSE_H + G_CUT_H) / 2))
                    {

    SDL_LogInfo(SDL_LOG_CATEGORY_APPLICATION, "MOUSE IS
KILLED");
                        SDL_RenderClear(renderer);
                        SDL_QueryTexture(pictures[0].texture,
(Uint32 *)&pixelFormat, (int *)&access, &texture_dimensions.w,
&texture_dimensions.h);
                        texture_dimensions.x = 0;
                        texture_dimensions.y = 0;
                        SDL_RenderCopy(renderer,
pictures[0].texture, &texture_dimensions, &texture_dimensions);
                        for (j = 0; j < 3; j++)
```

```
                        {
                            texture_dimensions.h = G_MOUSE_H;
                            texture_dimensions.w = G_MOUSE_W;
                            texture_dimensions.x = mxy[j][0];
                            texture_dimensions.y = mxy[j][1];
                            if (j == 1)
                            {
                                SDL_RenderCopy(renderer,
pictures[4].texture, NULL, &texture_dimensions);
                            }
                            else{
                                SDL_RenderCopy(renderer,
pictures[1].texture, NULL, &texture_dimensions);
                            }
                        }
                        texture_dimensions.h = G_CUT_H;
                        texture_dimensions.w = G_CUT_W;
                        texture_dimensions.x = cx - G_CUT_W / 2;
                        texture_dimensions.y = cy - G_CUT_H / 2;
                        SDL_RenderCopy(renderer,
pictures[2].texture, NULL, &texture_dimensions);
                        SDL_RenderPresent(renderer);
                    }
                }
            }
            if (event.type == SDL_QUIT)
                should_exit = 1;
            event_pending = 0;
        }
    }
    for (i = 0; i < num_pictures; i++)
        SDL_DestroyTexture(pictures[i].texture);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    for (i = 0; i < num_pictures; i++)
        SDL_FreeSurface(pictures[i].surface);
    SDL_free(pictures);
    IMG_Quit();
    SDL_Quit();
    return 0;
}
```

实验结果/结论

1． 实验结果

✓ 运行程序，程序进入全屏显示，先显示一张打地鼠的欢迎图片，然后切换为一片田野，有 2 只地鼠和一只老鼠随机的在田野中出现。点击任何一只鼠，则会出现锤子。

游戏界面

击中鼠之后的界面

2． 实验结论

- ✓ SDL2 默认只支持.bmp 图像(SDL_LoadBMP())，要想使用其它格式的图片，可在 http://www.libsdl.org/projects/SDL_image 页面下载 SDL_image 扩展组件，它支持 BMP,GIF,JPEG,LBM,PCX,PNG,PNM,TGA,TIFF,WEBP,XCF,XPM,XV 格式。SDL_image 扩展组件的一般使用流程为首先进行初始化

```
if (IMG_Init(IMG_INIT_PNG) == -1)
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Could not
initialize SDL IMG.");
        exit(-2);
    }
```

然后加载图像文件 IMG_Load，使用完毕执行 IMG_Quit()。